Contributions of Formal Language Theory to the Study of Dialogues

Maria Adela Grando

Supervisors: Dr. Victor Mitrana (University of Bucharest, University Rovira i Virgili)

Dr. Maria Dolores Jiménez López (University Rovira i Virgili)

2nd International PhD School in Formal Language and Applications

Research Group on Mathematical Linguistics (GRLMC)

Rovira i Virgili University (URV)

Tarragona, SPAIN

- 2009 -

Table of Contents

List of Figures

Abstract

For more than 30 years, the problem of providing a formal framework for modeling dialogues has been a topic of great interest for the scientific areas of Linguistics, Philosophy, Cognitive Science, Formal Languages, Software Engineering and Artificial Intelligence. In the beginning the goal was to develop a "conversational computer", an automated system that could engage in a conversation in the same way as humans do. After studies showed the difficulties of achieving this goal *Formal Language Theory* and *Artificial Intelligence* have contributed to *Dialogue Theory* with the study and simulation of machine to machine and human to machine dialogues inspired by *Linguistic* studies of human interactions.

The aim of our thesis is to propose a formal approach for the study of dialogues. Our work is an interdisciplinary one that connects theories and results in Dialogue Theory mainly from Formal Language Theory, but also from another areas like Artificial Intelligence, Linguistics and Multiprogramming.

We contribute to Dialogue Theory by introducing a hierarchy of formal frameworks for the definition of protocols for dialogue interaction. Each framework defines a transition system in which dialogue protocols might be uniformly expressed and compared. The frameworks we propose are based on finite state transition systems and Grammar systems from *Formal Language Theory* and a multi-agent language for the specification of dialogue protocols from *Artificial Intelligence*. Grammar System Theory is a subfield of Formal Language Theory that studies how several (a finite number) of language defining devices (language processors or grammars) jointly develop a common symbolic environment (a string or a finite set of strings) by the application of language operations (for instance rewriting rules). For the frameworks we propose we study some of their formal properties, we compare their expressiveness, we investigate their practical application in Dialogue Theory and we analyze their connection with theories of human-like conversation from *Linguistics*.

In addition *we contribute to Grammar System Theory* by proposing a new approach for the verification and derivation of Grammar systems. We analyze possible advantages of interpreting grammars as multiprograms that are susceptible of verification and derivation using the Owicki-Gries logic, a Hoare-based logic from the *Multiprogramming* field.

Acknowledgements

Writing this thesis could have been an impossible task if I would not have count with the support, advice and help from the people I would like to thank here.

Special thanks to Dr. Victor Mitrana for his patience towards me and for his impartial and experienced advices. Also thanks to Dr. Dolores Jiménez-López and Dr. Gemma Bel Enguix, who supported me from the beginning to the end giving me their desinterested advice and support.

I am very grateful to Dr. Carlos Martin Vide, director of the PhD School in Formal Languages and Applications, because he gave me the chance to join this doctorate program and provided me with advice when I needed it most. Also I am indebted with the Ministry of Education and Science of Spain for the economical support provided with the mobility grant “Becas de Movilidad en Programas de Doctorado 2003 (modalidad B)” and the four years grant “Formación de Profesorado Universitario” AP 2003-1709.

I am indebted to Dr. David Robertson for supervising my stays during 2005 and 2007 in the Center for Intelligent Systems and their Applications (CISA) in the University of Edinburgh (UK). I also want to thank Dr. Chris Walton who generously helped me with the research I undertook there. I am very grateful to Dr. David Glasspool for his support and comprehension during the last months of completion of the thesis. I can not forget to thanks the disinterested and generous help that I got from the CISA members Michelle Galea and Rónán Daly. To them and to all the people from the CISA group I am very grateful for a wonderful personal and academic experience.

All my gratitude to Dr. Simon Parsons and Dr. Betsy Sklar who supervised my 6 months stay during 2006 in the Department of Computer and Information Science at Brooklyn College, CUNY

(USA). They made me feel “like in home” and they generously shared with me their time and knowledge.

I also want to thank all the members of the “Research Group on Mathematical Linguistics” in Rovira i Virgili University and all the professors who visited the group and contributed to my training as researcher. In particular I want to mention Dr. Artiom Alhazov, Dr. Marc Bezem, Dr. Peter Cholak, Dr. Adrian Horia Dediu, Dr. Zoltán Ésiz and Dr. Yuri Rogozhin. Also I would like to thank all the PhD students in the group for so many nice academic and personal moments spent together. Specially to my dear friends: Vikas Kumar, Roslin Sagaya Mary, Alexander Perekrestenko, Alexander Krassovitskiy.

To Thomas French, for his love, support and understanding during the hardest part of this process.

Por último quiero agradecer a mi familia. A mi padre por seguir siendo mi guía. Y a mi madre y hermanos porque a pesar de la distancia han estado más cerca de mi que nunca. A mis queridas tías Graciela y Damiana por brindarme un hogar en Areyns de Munt.

To all of you THANK YOU.

Chapter 1

Introduction

1.1 Context, state of the art

Dialogue Theory has it roots in Aristotelian times (300 BC) but it became a very active research area in the 1970s with applications in various fields.

Linguistics has contributed to Dialogue Theory introducing and supporting the conjecture that it is not possible to get a complete simulation of human language. The main argument of this position focuses on the impossibility of formalizing the context needed for human-level language understanding, as explained in [Dre92]. Larsson's ideas in that regard are very interesting:

> 'If we accept the argument that 'the background is not formalizable' and that computers will never achieve human-level language understanding, does it follow that formal and computational research on dialogue and dialogue systems is useless? Of course not; it provides a great potential for improving on human-computer interaction. But granted this, has theories of human language use now been shown to be of no use to research on human-computer dialogue? Again, of course not. For one thing, if we want dialogue systems that are reasonably human-like in their behavior, these systems will need to be designed on the basis of theories of human language. [...] We may use these theories as providing important clues about how to best build dialogue systems.' [Lar05]

The conjecture that the background of human dialogues is not formalizable has produced a reformulation of the goals in Dialogue Theory. Currently one of the main goals of formal Dialogue Theory is the definition of effective human to machine and machine to machine goal-oriented dialogue systems. As technologies become more complex command-line or menu-based graphical interfaces become increasingly impractical. This justifies the introduction of human-machine interfaces based on intelligent conversational agents.

Formal Language Theory has contributed with different grammatical models to the development of the modules involved in human-machine interfaces. The usual modules in a human-machine interface are:

- Speech recognition module: takes an audio input and transforms it into a string of words.

- Natural language understanding module: produces a semantic representation which is appropriate for the dialogue task.

- Natural language generator: chooses what to say and how to say it to the user.

- Task manager: keeps information related to the domain of application of the interface.

- Dialogue manager: controls the architecture and structure of the dialogue. It takes input from the two first modules, maintains some sort of state, interfaces with the task manager and passes output to the natural language generator module.

The first interfaces for human to machine dialogues are database question-answering systems; see for instance [Woo67], [Woo73], [Wei66]. These systems consist on a machine asking questions to a user, interpreting what the user said, and retrieving the corresponding answer from a data base when the information is enough. These interfaces are system-initiative, the machine completely controls the conversation. The system asks the user questions ignoring (or misinterpreting) anything the user says that is not a direct answer to the machine's question and goes on with the next question. The dialogue structure of question-answering systems corresponds to sequences of pairs of questions followed by answers. The dialogue manager of these interfaces is implemented as *dialogue grammars or finite-state automata* where the states represent dialogue states and the speech acts become the labels of the transitions.

Because pure system-initiative dialogue architectures are too restricted, mixed initiative architectures are introduced: frame-based architectures, plan-based architectures, information state frameworks and dialogue state models. In mixed initiative architectures the conversational initiative can shift between the machine and the human.

Frame-based or form-based architectures rely on the structure of frames to guide the dialogue. The machine associates questions to gaps in a frame. Then the machine asks the user questions in order to fill in the gaps in the frame. When all the gaps in a frame are filled in the system can perform a data base query and return the result to the user. The user is allowed to guide the dialogue by giving extra information that fills in another gaps in the frame. If the user happens to answer more than one question at a time, the system must fill in the corresponding gaps and remember not to ask the user the questions associated with the gaps that were already filled in. Frame-based dialogue systems are only capable of limited domain-specific conversations. This is because the semantic interpretation and generation processes in frame-based dialogue systems are based only on what is needed to fill in the gaps. Examples of frame-based interfaces are [CC00] and [SP00].

More flexible than frame-based architectures are the plan-based systems [CP79]. Developed in *Artificial Intelligence* field, plan-based systems are based on the idea of constructing plans to generate and interpret sentences. On these systems the machine should come up with a plan for getting from the interlocutor the information needed to perform a data base query. Besides, the machine can interpret a speech act by running the planner in reverse, using inference rules to infer what plan the interlocutor had to say what he said. Some examples of plan-based architectures can be found in [AFS01] and [LRG06].

Other mixed-initiative approach is the one based on the view of dialogue management functions in terms of information states. Information states are more complex representations of the dialogue context than the ones that can be defined with static states in dialogue grammars and forms in frame-based systems. According to [TL03] an information state framework consists of:

- Informational components. Corresponding to the common context (obligations and commitments, linguistic and intentional structure, shared knowledge) and internal motivating factors (beliefs, intentions, user models).
- Formal representations of informational components. Sets, strings, lists, typed feature structures, record, expressions in some logic or any arbitrary data types can be used to represent informational components.
- Dialogue moves. Set of external stimulus, in general natural language utterances, that trigger the updating process of information states.
- Update rules. According to the information state, the history of dialogue moves, and the update strategy, some of these rules are selected to modify the information state.
- Update strategy. Function that determines the criteria to select updating rules. It can be as simple as to "select an arbitrary rule" or as complex as to involve probabilistic studies.

Information states frameworks are a very generic and abstract notion that can embrace a broad class of dialogue transition systems. So far information state models have not been formally defined. Any dialogue transition system where the previously mentioned components can be found defines a dialogue move engine. Examples of dialogue system toolkits that allow the definition of dialogue move engines are mentioned in [BCEL99].

With respect to dialogue state frameworks, according to [LT00] it differs from information state models in the following way:

> “It is important to distinguish information state approaches to dialogue modeling from other, structural, dialogue state approaches. These latter approaches conceive a “legal” dialogue as behaving according to some grammar, with the states representing the results of performing a dialogue move in some previous state, and each state licensing a set of allowable next dialogue moves. The "information" is thus implicit in the state itself and the relationship it plays to other states. It may be difficult to transform an information state view to a dialogue state view, since there’s no necessary finiteness restriction on information states [...] and the motivations for update and picking a next dialogue move may rely on only a part of the information available, rather than to the whole state. On the other hand, it is very easy to model dialogue state as information state: the information is the dialogue state, itself. [...], the dialogue moves will be the same moves that are used in the dialogue state framework, the update rules will be the transitions in the dialogue state model, formulated as an update to a new state, given the previous state and performance of the action, and the update strategy will be much the same as in the transition network.”

The formal dialogue models introduced in [AJR01] and [JL00, CVJLMV99] are information state frameworks. Both works are based on Grammar Systems Theory, a subfield of Formal Language Theory. Informally a Grammar system consists of several grammars that work in a distributed and cooperative way contributing to generate or accept a language. In [JL00, CVJLMV99] and [AJR01] grammars were interpreted as speakers and a shared dialogue context was provided. Both approaches focused on the dialogue structure avoiding the problems of human speech recognition and synthesis (voice recognition, representation of semantics, classification of speech acts, etc). Besides in [JL00, CVJLMV99] they presented linguistic evidence that Grammar systems provide many of the structural design and behavioral features that human speech has: coordination, cooperation, interaction, dynamism, flexibility, emergence, coherence, copresence, contextuality, audibility, instantaneity, evanescence, simultaneity, extemporariness and turn-taking. Whether this framework is practical has not been investigated yet.

Within *Computer Science*, Dialogue Theory has recently been taken up as a broader subject of investigation. From the 80's computer scientists are getting seriously interested in Dialogue Theory, and even seeing it as a necessary part of their research initiatives in fields like expert systems technology, multi-agent systems, robotics and argumentation-based systems.

The growing field of expert systems provides a natural application for Dialogue Theory. In an expert system a user of advice or information consults an expert source (human or machine). During the dialogue the user asks questions that the expert answers. Another application of Dialogue Theory are the multi-agent systems. Communication in multi-agent systems is crucial to allow agents to collaborate and share their partial knowledge in order to solve complex problems that otherwise they could not solve in isolation. Agent communication languages like [PFPS+92][BS96] [Woo99] and dialogue protocols of the type of [Vas04, ERS+01] [Wal04c] have been introduced for message and knowledge exchange between agents.

With respect to *Argumentation Theory*, according to [vEGH96b]:

> "Argumentation is a verbal and social activity of reason aimed at increasing (or decreasing) the acceptability of a controversial standpoint for the listener or reader, by putting forward a constellation of propositions intended to justify (or refute) the standpoint before a rational judge."

Since its start in the middle 80's Argumentation Theory has found a wide range of applications in both theoretical and practical branches of Artificial Intelligence and Computer Science [vEGH+96a]. In particular in multiagent systems argumentation theories provide a powerful framework for defining the way agents get engage in deliberations, resolve conflicts interchanging arguments in favor or against some conclusion, negotiate, shared data and knowledge and take decisions based on potentially incomplete or inconsistent information. In recent years the study of Argumentation Theory in the context of dialogues has been quite intensive -including that of Kraus [KNS93], Maudet [ME98], McBurney [McB02], Reed [Ree98], Schroeder et al. [SPR98], Sycara [Syc89] and Parsons [PWA02] [PWA03a] [TP05].

It seems that *Dialogue Theory*, although it has ancient roots, has recently being regarded as an important subject of research with multiple applications and potential uses in different areas of study.

In this thesis we analyze, mainly from the perspective of Formal Language Theory, some aspects related to the dialogue interaction. But we also look for connections and contributions between this subject and fields like Artificial Intelligence, Linguistics, Multiprogramming and Argumentation Theory. Therefore our approach can be framed in the area of Dialogue Theory and be considered an interdisciplinary work.

1.2 Overview and main contributions

Our work is an interdisciplinary one. *We contribute to Dialogue Theory* introducing a hierarchy of formal frameworks in which dialogue protocols might be uniformly expressed and compared. Some of the frameworks we define are based on finite state transition systems, while others are based on Grammar systems.

While in *Dialogue Theory* sets of desirable dialogue features are used to compared the frameworks used for specifying dialogues, in *Formal Language Theory* the usual approach when defining a new formal framework is to analyze its connection with other existing grammatical systems in order to compare expressiveness and complexity results. Here we follow the later approach comparing the expressiveness of the frameworks we define and studying some of their formal properties.

For the frameworks that we introduce we analyze their expressive power in terms of the Chomsky hierarchy and in terms of features like: number of speaker, the way agents perceive the context in order to take decisions (perception in blocks or scattered, length of the context they can perceive), etc. Besides we mention the practical use of the frameworks for the specification of dialogues and we compare them with respect to more informal properties considered in the area of Dialogue Theory, which are:

- restrictions on the maximum number of speakers,
- capacity of the agents to learn or modify their own knowledge bases or the shared knowledge base through the dialogue,
- restrictions over shared memory (memory seen as an unrestricted record of dialogue history, memory seen as a frame of questions to be answered, memory restricted to the last uttered locution, etc)
- backtracking (the capacity to reply to locutions uttered at any earlier step of the dialogue and not only the previous one),
- turn-taking rules used (the way that agents shift turns to talk),
- policies for replying (multiple replies or a unique reply),
- flexibility in the selection of locutions used,
- the type of agent knowledge bases allowed,
- the complexity of agent reasoning strategies,
- allowance of interchange of arguments, contraarguments and implementation of agent reasoning strategies with incomplete or inconsistent knowledge bases,

- dialogue initiative (system initiative, user initiative, mixed initiative),
- provision of features that, according to linguistic theories of Dialogue Theory, are close-to-human conversation.

Our intention is to *contribute to Dialogue Theory* providing the designers of dialogue protocols with a hierarchy of formal frameworks from where they can choose the model that suits better their requirements of expressiveness minimizing computational complexity and undecidability of results.

With the hierarchy we introduce *we also contribute to Formal Language Theory* with the development of grammatical frameworks to define dialogue protocols. Some of the frameworks we propose are based on Grammar system variants, and others are defined as finite state transition system. Those frameworks that we propose as Grammar systems variants can be seen as the continuation of the models presented in [JL00, CVJLMV99] and [AJR01]. From them we took the idea of defining a formal framework for the specification of dialogue protocols as a Grammar system variant where: each population $\mathcal{A}_i$, $1 \leq i \leq n$, represents a role that agents can play in the dialogue, each grammar A_j belonging to population $\mathcal{A}_i$ is interpreted as a speaker who participates in the conversation as described by role $\mathcal{A}_i$, and the message interchange is done through a shared conversational context.

In particular we introduce *Conversational Extended Reproductive Eco-Grammar (ConvEREG)* systems as variants of Reproductive EG system where we replace rewriting rules for strings for describing the speakers behavior. In our model each speaker has a string description of his expected behavior during the conversation. This string description is a protocol definition in a Multi-Agent Protocol calculus that we developed for this purpose and that we called MAP^a. While strings can be changed during run-time according to the dialogic state, rewriting rules are defined at design time and they remain fixed during all the conversation. Since our framework does not have this restriction it can be applicable to define dialogue protocols whose dialogic space is not known beforehand but emerges from the process of communication and from the agent knowledge. Another difference of *ConvEREG* systems with respect to previous approaches [JL00, CVJLMV99] and [AJR01] is that we consider agents(speakers) as "grey boxes" whose inner implementation details are unseen. We specify only those agent details that are directly affected by the interaction with the environment and that determine their observable behavior. The main advantage of our approach is that in case of dialogues involving agents that are not fully knowable or whose complexity of representation is hight, we can still construct an abstract representation and reason about the dialogue interaction.

ConvEREG systems correspond to information state theory and can be considered an improvement with respect to previous grammatical approaches defined for the simulation of dialogues, because they are:

- *Focused on dialogue structure and not concerned with natural language processing.*
- *An abstract architecture for specifying agent observable behavior.* In this way we avoid the representation of agents with complex internal behaviors and we can still define and study dialogues and interactions where some or all the agents cannot be fully knowable, like for instance legacy components or open systems.
- *Generic, highly expressive, dynamic, emergent, reusable, modular, flexible and adequate for applicative use.*

- *Linguistically well founded.* Because a *ConvEREG* is defined as a Grammar system variant made up of a community of evolving agents that interact with each other and with their shared evolving environment. According to [CVJLMV99]:

  > "This is similar to what happens if people participate in a conversation. The *interlocutors* of the conversation correspond to the *agents* and the shared *context*, that is, the speech and the knowledge on the topic of the conversation, corresponds to the *environment*. Evolution and actions, together, realize the modification of the context."

  Therefore this framework exhibits some features characteristic of human speech like copresence, contextuality, audibility, instantaneity, evanescence, simultaneity, extemporariness and emergent behavior.

- *The combination of different approaches developed for the study of dialogues.* In particular the MAP^a language that we define for describing cooperation and communication in Multi-agent systems in *Artificial Intelligence*, the Conversational Grammar systems [JL00] from *Formal Language Theory* and studies of human dialogues from *Linguistics*.

Because *Extended ConvEREG* (*EConvEREG*) systems prove to have the same expressive power as Turing Machines, we introduce some restrictions to decrease their expressive power getting two subclasses: one based on Eco-Grammar systems that we called *Conditional Eco-grammar (CondEG)* system, and other based on Reproductive Eco-Grammar systems that we called *regularly Controlled Reproductive Eco-grammar (rCREG)* system. *Extended CondEG* (*ECondEG*) systems with non erasing rewriting rules prove to be a subclass of Context Sensitive (CS) grammars. While *Extended rCREG (ErCREG)* systems prove to be a type of mildly CS grammar.

Based on finite state transition systems we introduce the *Conversational Finite State Transition* (*ConvFST*) systems to formally specify dialogue protocols between a fixed number of $n \geq 1$ agents. Each agent A_i, $1 \leq i \leq n$, is provided with a private knowledge base K_i containing its beliefs. These knowledge bases are fixed during the dialogue; this means that the agents do not learn through the dialogue. The dialogue moves are described by a finite state transition system where the states correspond to stages in the conversation and transitions are conditional and labeled by locutions from a locution set. For a labeled transition to be triggered the precondition associated with the locution that labels it must be satisfied. And the preconditions of the locutions are computable function, not necessarily complete, over some logic which are evaluated according to the agent knowledge and shared knowledge. If a transition is triggered a new state is reached and some operation can be performed over a string of shared knowledge. This string of shared knowledge records the information exchanged and the knowledge learned by the agents during the dialogue. This framework is very flexible, it allows the definition of dialogue protocols with: backtracking (the capacity to reply to locutions uttered at any earlier step of the dialogue and not only the previous one), different turn-taking rules (agents can make several moves before the turn shifts), arbitrary sets of locutions, arbitrary type of agent knowledge bases and reasoning strategies, multiple replies (indeterministic transition systems).

ConvFST systems correspond to dialogue state theory and they are a variant of finite state transition system. We have shown the suitability of this framework to simulate the argumentation-based dialogue systems introduced in [Amg98]. *EConvFST* systems prove to have the same expressive power as Turing Machines, therefore we introduce three subclasses of the *ConvFST* systems where we fix some restrictions to decrease their expressive power: *Conditional Finite State Transition* (*CondFST*) systems, where *Extended CondFST (ECondFST)* systems with non erasing rules are a subclass of CS grammars, *limited memory Finite State Transition* (*lmFST*) systems where *Extended lmFST (ElmFST)* systems the same expressive power as CS grammars and *regularly Controlled Finite State Transition (rCFST)* systems, where *Extended rCFST (ErCFST)* systems are a type of mildly CS grammar. We also show the suitability of *rCFST* systems to simulate frame-based mixed initiative dialogues.

The frameworks that we mentioned above are variants of finite state transition systems and can be seen as the continuation of the models presented in [Vas04, ERS$^+$01] [HK98][MW97][PC96].

Although the main aim of this thesis is to explore possible ways to apply Grammar System Theory to the formal study of dialogues, *we also contribute to Grammar System Theory*. We do it by addressing the problem of formally proving that a Grammar system verifies certain properties or that it produces a desired outcome. In the programming field there are two ways to prove that a program satisfies a specification: verify that some existing program meets its specification, or simultaneously derive a program and the proof of satisfaction of the specification given. In Grammar systems mainly analysis by cases is used as verification strategy and so far no formal derivation strategy has been tried. Inspired in the programming field we propose a new approach for the formal verification and derivation of Grammar systems. Our proposal consists on interpreting Grammar systems as programs in order to make use of the formal verification and derivation strategies available in the programming field.

1.3 Structure of the thesis

This thesis is divided in six chapters.

Chapter 2: Grammar systems
We introduce Grammar System Theory and we explain the Grammar system models that are used through the thesis.

Chapter 3: A new approach for the verification and derivation of Grammar systems
We show that it is possible to interpret some Grammar system variants (Cooperating Distributed Grammar systems, Parallel Communicating Grammar systems, Parallel Communicating Grammar systems with communication by command and Eco-Grammar systems) as multiprograms. We explain and exemplify the use of a Hoare-based logic, called Owicki-Gries Theory, over those Grammar systems that can be interpreted as multiprograms. We introduce Owicki-Gries Theory as an alternative approach to analysis by cases, to verify the satisfaction of properties in Grammar systems. We explain a novel idea consisting of using Owicki-Gries Theory to simultaneously construct Grammar systems and the proof that they satisfy a property.

Through the next two chapters we show the applicability of Formal Language Theory, and Grammar System Theory in particular, to the development of a hierarchy of formal frameworks for the modelization of dialogue protocols. For the frameworks that we define in both chapters we study mainly their expressive power and the effect that the restrictions that we impose on them have over their applicability in the modelization of dialogue protocols: which features considered in Dialogue Theory as desirable are lost when we restrict the expressive power of the framework we introduce.

Chapter 4: Using Grammar systems for specifying dialogue protocols
We define *Enlarged Reproductive Eco-Grammar (EREG)* systems, a new variant of Reproductive Eco-Grammar systems. We formally introduce *Conversational Enlarged Reproductive Eco-Grammar (ConvEREG)* systems as EREG systems where the agent behavior is not described by rewriting rules, but by process descriptions in a language that we define for this purpose: the Multi-Agent Protocol (MAP^a) calculus. Under the restriction that the agents can only invocate computable functions, not necessarily complete, it turns out that *EConvEREG* systems have the same expressive power as Turing Machines, therefore we impose restrictions in order to reduce their expressive power. We define two subclasses of *ConvEREG*: *Conditional Eco-Grammar* (*CondEG*) systems and *regularly Controlled Reproductive Eco-Grammar (rCEREG)* systems.

CondEG systems are a variant of Eco-Grammar system where each agent's private knowledge base is restricted to two finite sets of conditional rules. In each derivation step every agent inspects its two sets of conditional rules in order to decide its participation in the conversation and how to change its mental state. The first set of rules is used by the agent to decide how to change the dialogue context (environment) according to the presence and\or absence of information in its mental state (agent state). The second set of rules is used by the agent to change its mental state according to the presence and\or absence of information in the dialogue context. *ECondEG* systems with non erasing rules are a subclass of CS grammars.

rCEREG systems correspond to a variant of Reproductive Eco-Grammar system to which we attach a turn talking policy given by a regular set that determines the order that speakers must take to participate in the dialogue. Under the restriction that agents can only invocate computable functions, it turns out that *ErCEREG* systems have the same expressive power as *regularly Controlled (rC)* grammars with non-erasing CF rules [GS68][DPS97]. Class *rC* corresponds to a type of mildly CS grammar where the membership problem is decidable, the emptiness problem is NP-hard and the finiteness problem is NP-hard.

Chapter 5: Using finite state transition system for specifying dialogue protocols
We devote this chapter to the definition of frameworks based on a finite state transition system for the definition of dialogue protocols. First we introduce the so-called *Conversational Finite State Transition (ConvFST)* system. *ConvFST* systems allow the specification of dialogue protocols where the number of agents is fixed, agents are provided with fix sets of private believes, the interaction model is described by a finite state transition system whose transitions are conditionals and labeled with locutions, and the shared knowledge is saved in a string whose content can be modified. The states of the automata correspond to possible stages of the conversation, and the transitions to dialogue moves. For a labeled transition to be triggered the precondition associated with the locution that labels it must be satisfied. The preconditions of the locutions are formulas over some logic which are evaluated according to the agent's knowledge and shared knowledge. Checking the satisfaction of the locution's precondition is computable. If a transition is triggered

a new state is reached and some operation can be performed over the shared knowledge. After showing the applicability of *ConvFST* systems in Dialogue Theory, and in Argumentation Theory in particular, we analyze its expressiveness. We discover that *Extended ConvFST* (*EConvFST*) systems have the same expressive power as *EConvEREG* systems, therefore we define three subclasses of *ConvFST*: *Conditional Enlarged Finite State Transition (CondFST)* systems, *limited memory Finite State Transition* (*lmFST*) systems and *regularly Controlled Finite State Transition (rCFST)* systems.

In *CondFST* systems the agents are provided with finite sets of conditional rules that they use to decide their participation in the dialogue. The agents use these rules to decide how to modify the string corresponding to the shared knowledge depending on the presence and\or absence of substrings(information) in that string. *Extended CondFST (ECondFST)* systems with non erasing rules are a subclass of CS grammars.

In *lmFST* systems the knowledge shared by the agents is limited to at most the last uttered locution and the decision procedures used by the agents to determine when to utter a locution are decidable. Because there are no restrictions over the locutions used very complex dialogues can be modeled, where locutions can contain information of previously uttered locutions and even the whole dialogue history. *Extended lmFST (ElmFST)* systems have the same expressive power as CS grammars.

In *rCFST* systems the locutions have no parameters and each locution is associated with a CF rule such that the locution's precondition consists on checking if the associated CF rule can be applied over the string of shared knowledge. If the precondition associated with a locution is satisfied, then the locution is uttered and a new string of shared knowledge is obtained from the application of the CF rewriting rule associated with that locution. *Extended rCFST (ErCFST)* systems have the same expressive power as *regularly Controlled* grammars and therefore correspond to a type of mildly CS grammar.

Finally **Chapter 6: Conclusions and Future Work.**

Chapter 2

Grammar systems

2.1 Introduction

Grammar systems [CVD90] were introduced as a new subfield of Formal Language Theory. They were developed to provide a formal framework for the abstract representation of computing models from a new paradigm which was born in the 80s based on distribution, decentralization, emergent behavior, parallelism, modularity and communication. The new paradigm tried to computationally reflect the observation of real-life features like the reproduction of living beings, the behavior of molecules, the activity of the brain, different ways of communication, etc. This paradigm introduced a new way of thinking about solving problems and it proved to increase the efficiency in the time to find the solutions. Following these ideas, different computing strategies were developed in robotics, artificial intelligence, computer science, etc. While parallel models have proved to be more powerful than those traditionally used, their main disadvantage has been, in general, their lack of theoretical foundations. Take for example the case of genetic algorithms from the area of Artificial Intelligence, as we pointed out in [DG05a] and [DG05b]. Grammar systems were the grammatical counterpart of this new computing paradigm and they provided a new approach different from the traditional one in Formal Language Theory. While before one language was generated by one grammar or generating mechanism, Grammar systems consist on several grammars that work in a distributed and cooperative way to contribute to generate or accept a language. According to [CVD90] all Grammar system variants, independently of their type, share a common definition:

> "A Grammar system consists of several (a finite number) of language determining devices (language processors) which jointly develop a common symbolic environment (usually, a string or a finite set of strings) by applying language theoretic operations to it. The symbolic environment can be shared by the components of the system or it can be given in the form of a collection of separated sub-environments, each belonging to a language processor. At any moment of time, the state of the system is represented by the current string describing the environment (the collection of strings of the sub-environments). The functioning of the system is realized by changes of its states. Depending on the variant of multi-agent systems which is represented by the actual Grammar system, in addition to performing derivation steps, the language processors are allowed to communicate with each other. Usually, this is done by exchange

of strings which can be data (for example, sentential forms in derivation) or programs (productions or coded form of some operation). The behavior of the Grammar system can be characterized by the set of sequences of environmental states following each other starting from an initial state or by the set of all states of the environment or that of a sub-environment which originate from the initial state and satisfy certain criteria. The second case defines the language of the system."

Since its creation, Grammar systems have proven to be a flourishing area. Different variants of Grammar systems have been introduced, their computational power and grammatical properties have been studied and compared, and some empirical applications have been found.

Section 2.2 starts with some preliminary basic knowledge of Formal Language Theory. Section 2.3 gives the Grammar system definitions that will be used during the rest of the work. Finally section 2.3.6 mentions some Grammar systems definitions that have been successfully introduced for applicative use.

2.2 Preliminaries

Prior to the formal definition of Grammar systems, we fix some basic notation and concepts from Formal Language Theory. For all unexplained notions the reader is referred to [RS97].

An *alphabet* is a finite and nonempty set of symbols. Any sequence of symbols from an alphabet V is called a *string (word)* over V. The set of all strings over V is denoted by V^* and the empty string is denoted by λ. Further, $V^+ = V^* \setminus \{\lambda\}$. The number of occurrences of a symbol $a \in V$ in a word $w \in V^*$ is denoted by $(w)_{\#a}$ and the length of w is denoted $\mid w \mid$.

A Chomsky grammar is a quadruple $G = (N, T, S, P)$, where N is the nonterminal alphabet, T is the terminal alphabet, $S \in N$ is the axiom, and P is the (finite) set of rewriting rules. The rules are presented in the form $u \rightarrow v$ and used in derivations as follows: $x \Longrightarrow y$ is written if $x = x_1ux_2$, $y = x_1vx_2$ and $u \rightarrow v$ is a rule in P (one occurrence of u in x is replaced by v and the obtained string is y). Denoting by $\Longrightarrow^*$ the reflexive and transitive closure of $\Longrightarrow$, the language generated by G is defined by:

$$L(G) = \{x \in T^* \mid S \Longrightarrow^* x\}.$$

A Chomsky grammar can be:

- Regular, if it is a right linear grammar or a left linear grammar.
- Right linear, if every production is presented in the form: $A \rightarrow a$, $A \rightarrow aB$ where $A, B \in N$ and $a \in T^*$.
- Left linear, if every production is presented in the form: $A \rightarrow a$, $A \rightarrow Ba$ where $A, B \in N$ and $a \in T^*$.
- Linear, if it has at most one non terminal on the right hand side of any production.
- Context free, if every production has the shape: $A \rightarrow x$ where $A \in N$ and $x \in (N \cup V)^*$.

- Context sensitive (CS), if every production has the shape: $\alpha A\beta \rightarrow \alpha\gamma\beta$ or $S \rightarrow \lambda$ where $A \in N$, $\alpha,\beta \in (N \cup T)^*$, $\gamma \in (N \cup T)^+$ and S is not in the right-side of any production.
- Type-0 or arbitrary Chomsky grammar, if it is one without any restriction over its productions.

The families of languages generated by regular, left linear, right linear, linear, context free, context sensitive and type-0 Chomsky grammars are denoted by REG, LL, RL, LIN, CF, CS, RE, respectively. The family of finite languages is denoted by FIN.

Similarly a 0L system (an interactionless Lindenmayer system) is a triple $H = (V, P, w)$ where V is a finite alphabet, P is a set of context free rules over V and $w \in V^*$ is the axiom. Moreover P has to be complete, that is, for each symbol a in V there must be at least one rule $a \rightarrow x$ in P with this letter a on the left-hand side. $0L$ systems use parallel derivations: it is said that x directly derives y in a 0L system $H = (V, P, w)$ with $x, y \in V^*$, written as $x \Rightarrow_H y$, if $x = x_1x_2...x_n$, $y = y_1y_2...y_n$ where $x_i \in V$, $y_i \in V^*$ and the rules $x_i \rightarrow y_i$ are in P for $1 \leq i \leq n$.

A 0L system $H = (V, P, w)$ is propagating or P0L system if all the productions in P are λ-free.

A 0L system whose alphabet is divided into a nonterminal and a terminal alphabet $H = (N, T, P, w)$ is called an E0L system (an extended 0L system) and its language is defined by $L(H) = \{x \in T^* \mid w \Rightarrow^* x\}$ where the derivation takes places as in a 0L system.

A T0L system (a tabled 0L system) is a tuple $H = (V, P_1, ..., P_n, w)$, where each triple (V, P_i, w) is a 0L system; a string x derives a string y if $x \Rightarrow y$ with respect to some 0L component scheme (V, P_i). Extended versions can be defined for T0L systems too. They are called ET0L systems. We denote by 0L, E0L,T0L, ET0L the families of languages generated by 0L, E0L, T0L, ET0L systems, respectively. The following basic relations have been proved:

$$\begin{gathered} FIN \subset REG \subset CF \subset CS \subset RE, \\ 0L \subset T0L \subset ET0L, \\ CF \subset E0L \subset ET0L \subset CS, \\ CF \text{ is incomparable with both } 0L \text{ and } T0L \end{gathered}$$

We denote as **N** and **Z** the set of natural and integer numbers, respectively.
We consider:
$2^A = \{X \mid X \subseteq A\}$,
$card(A)$ the cardinality of set A, i.e. the number of elements of A,
$max(A) = x \leftrightarrow (\forall y \in A : x \geq y)$, where A is a set of numeric elements, is the maximal element in A,
$size(\alpha)$, where α is a stack of elements of type T, is the number of elements saved in α.

2.3 Main Grammar system models

2.3.1 Cooperating Distributed Grammar systems

The first and simplest Grammar system variant was introduced in [CVD90]. It was inspired by a model already used in Artificial Intelligence, the so-called *blackboard model* of problem solving. The idea is that there is a teacher that writes a problem on a common blackboard and all the students contribute individually, in turn, to the solution of the problem. The students only communicate through the blackboard, they cannot chat between themselves about the problem.

CD Grammar systems model the syntactic aspects of this problem: a CD Grammar system is a finite set of (usually generative) grammars which cooperate in deriving words of a common language. At any moment of the generation process there is exactly one sentential form. The component grammars generate the common string in turns, under a cooperation protocol, called the derivation mode. Formally:

Definition 1 *(Cooperating Distributed Grammar system) A CD Grammar system* Γ *of degree n,* $n \geq 1$, *and derivation mode f, is a construct*

$$\Gamma = (T, G_1, G_2, ..., G_n, S)$$

where:

- T is the terminal alphabet,
- S is the start symbol and
- $G_i = (N_i, T_i, P_i, f_i),\ 1 \leq i \leq n$

  where:

  - N_i is the nonterminal alphabet of G_i,
  - T_i is the terminal alphabet of G_i,
  - P_i is the production set of G_i and
  - $f_i \in \{t, *\} \cup \{= k, \leq k, \geq k \mid k \geq 1\}$ is the mode of derivation. According to the value it takes it limits the way the grammars work on the sentential form:

    * t-mode: $x \overset{t}{\underset{P_i}{\Longrightarrow}} y$ iff $x \overset{*}{\underset{P_i}{\Longrightarrow}} y$ and there is no $z \in (N \cup T)^*$ with $y \underset{P_i}{\Longrightarrow} z$,.

      For this mode of derivation the grammar contributes to the solution of the problem as much as it can.
    * $= k$-mode: $x \overset{=k}{\underset{P_i}{\Longrightarrow}} y$ iff there are $x_1, ..., x_{k+1} \in (N \cup T)^*$ such that $x = x_1$, $y = x_{k+1}$,

      and for each $1 \leq j \leq k$ $x_j \underset{P_i}{\Longrightarrow} x_{j+1}$.

      In this case, the contribution of the grammar is fixed to k successive derivation steps.
    * $\leq k$-mode: $x \overset{\leq k}{\underset{P_i}{\Longrightarrow}} y$ iff $x \overset{=q}{\underset{P_i}{\Longrightarrow}} y$ for some $q \leq k$.

      In this case a restriction of less than k contributions is provided.

- $\geq k$-mode: $x \underset{P_i}{\overset{\geq k}{\Longrightarrow}} y$ iff $x \underset{P_i}{\overset{=q}{\Longrightarrow}} y$ for some $q \geq k$.
  In this mode a minimal contribution from the agent is required, not less than k derivation steps.
- The $*$ mode of derivation, describes the case when the agent performs derivations on the world as long as it wants.

In the architecture of a CD Grammar system one can recognize the structure of the *blackboard model*. Each grammatical component G_i can be seen as a student. The axiom S is the formulation of the initial problem given by the teacher, and all the students (grammars) contribute to the solution of the problem by performing rewriting rules. While nonterminals N_i are seen as questions introduced by agent G_i, terminals T_i are the answers it provides. Finally, a solution is obtained when a sentential form w containing no question is obtained. At this point, w is a string of characters from the terminal set T.

Definition 2 *(Language) The language generated by a CD Grammar system* Γ *with degree n and derivation mode* $f \in \{*, t\} \cup \{\leq k, = k, \geq k \mid k \geq 1\}$ *is defined as:*

$$L_f(\Gamma) = \left\{ \begin{array}{l} w \in T^* \mid S \underset{P_{i_1}}{\overset{f}{\Longrightarrow}} w_1 \underset{P_{i_2}}{\overset{f}{\Longrightarrow}} \ldots \underset{P_{i_m}}{\overset{f}{\Longrightarrow}} w_m = w, \\ m \geq 1, 1 \leq i_j \leq n \end{array} \right\}$$

A component G_i may start working (gets enabled) on a sentential form w whenever w contains an occurrence of the left-hand side of a production from P_i. Which of the enabled components gets the current sentential form is decided in a nondeterministic way. The moment when the grammar G_i stops working on w is determined by the stop condition f_i.

The Grammar system is *homogeneous* when all its components work in the same mode and *hybrid* when each component can use different basic derivation modes.

We denote by $CD_n(f)$ the family of languages generated by homogeneous Cooperating Distributed Grammar systems with at most n components working in the f mode of derivation, where:

- $n \in \mathbf{N} \cup \{*\}$, where $*$ means that the number of components is irrelevant and
- $f \in \{t, *\} \cup \{\leq k, = k, \geq k \mid k \geq 1\}$.

2.3.2 Parallel Communicating Grammar systems

Parallel Communicating (PC) Grammar systems were introduced in [PS89] as a grammatical representation of the so-called *classroom model* of problem solving, which is a modification of the blackboard model. Here each student is given a problem written in his own notebook. Each student works on his own problem, but can ask for help from other students in the classroom in order to solve the problem.

Definition 3 *(Parallel Communicating Grammar system) A PC Grammar system of degree $n, n \geq 1$, is an $(n+3)$-tuple*

$$\Gamma = (N, K, T, (P_1, S_1), (P_2, S_2), (P_3, S_3), ..., (P_n, S_n))$$

where:

- N is a nonterminal alphabet,
- T is a terminal alphabet,
- $K = \{Q_1, Q_2, ..., Q_n\}$ (the sets N, T, K are mutually disjoint) and
- P_i is a finite set of rewriting rules over $N \cup K \cup T$, and $S_i \in N$, for all $1 \leq i \leq n$.

Let $V_\Gamma = N \cup K \cup T$. The sets P_i, $1 \leq i \leq n$, are called the components of the system, and the elements $Q_1, Q_2, ..., Q_n$ of K are called query symbols, the index i of Q_i points to the component P_i of Γ. An n-tuple $(x_1, x_2, ..., x_n)$ with $x_i \in V_\Gamma^*$ for all i, $1 \leq i \leq n$, is called a *configuration* of Γ. A configuration $(x_1, x_2, ..., x_n)$ directly yields other configuration $(y_1, y_2, ..., y_n)$ if either:

- No query symbol appears in $x_1, x_2, ..., x_n$ and then a componentwise derivation occurs, $x_i \Longrightarrow y_i$ in each component P_i, $1 \leq i \leq n$ (one rule is used in each component P_i), except for the case when x_i is terminal, $x_i \in T^*$; then $x_i = y_i$, or

- Query symbols occur in some x_i. Then a *communication step* is performed: every x_i (containing query symbols) is modified by substituting x_j for each occurrence of a query symbol Q_j, providing x_j does not contain query symbols. After all words x_i have been modified, the component P_j continues its work on the current string (in the *non-returning* case) or resumes working from its axiom (in the *returning* case). The communication has priority over the effective rewriting: no rewriting is possible as long as at least one query symbol is present. If some query symbols are not satisfied at a given moment, then they have to be satisfied as soon as other query symbols have been satisfied.

If only the first component is entitled to introduce query symbols, then the system is called *centralized,* otherwise it is called *non-centralized.*

Definition 4 *(Language) Given a PC Grammar system Γ the language generated by Γ is given by the following expression:*

$$L(\Gamma) = \{x \in T^* \mid (S_1, S_2, ..., S_n) \Longrightarrow (x, \alpha_2, ..., \alpha_n), \alpha_i \in V_\Gamma^*, 2 \leq i \leq n\}.$$

Hence, one starts from the n-tuple of axioms, $(S_1, S_2, ..., S_n)$, and proceeds by repeating rewriting and communication steps, until the component P_1 produces a terminal string. The component P_1 is called the *master* of the system.

We denote by $PC_n(Y)$ the family of languages generated by non-centralized returning Parallel Communicating Grammar systems with at most n components, each component with productions of type Y, where:

- $n \in \mathbf{N} \cup \{*\}$, where symbol $*$ refers that the parameter is irrelevant and

- $Y \in \{FIN, REG, CF, CS, RE\}$.

When the PC Grammar system is centralized, non-returning and non-returning centralized the prefixes C, N and NC, respectively, are added.

In PC Grammar systems the communication is on request. This means that communication takes place when an agent requests information from other agents in the network by introducing a query symbol. But there is also another way of communication, called communication by command, which was introduced as part of the WAVE paradigm [Err93]. Parallel Communicating Grammar systems with communication by Command (CCPC) were introduced in [CVKP96] to provide a syntactic model for this type of communication. Here each grammar in the system works on its own sentential form and it has one language playing the role of a filter. When the rewriting time is interrupted agents communicate by sending their sentential form to those agents whose filter allow them to receive that sentential form.

Definition 5 *(Parallel Communicating Grammar system with Communication by Command) A CCPC* Γ *is a construction of the form:*

$$\Gamma = (N, T, (S_1, P_1, R_1), (S_2, P_2, R_2), ..., (S_n, P_n, R_n))$$

where:

- N is the nonterminal alphabet,
- T is the terminal alphabet and
- (S_i, P_i, R_i), $1 \leq i \leq n$ are the components of the systems

  where:

  - $S_i \in N$ is the axiom,
  - P_i is the set of production rules over $(N \cup T)^*$ and
  - $R_i \subseteq (N \cup T)^*$ is the selector language of the component i.

A *rewriting step* in Γ is defined by $(x_1, ..., x_n) \Longrightarrow (y_1, ..., y_n)$ iff for each i, $1 \leq i \leq n$, $x_i \Longrightarrow^* y_i$ in P_i and there is no $z_i \in (N \cup T)^*$ such that $y_i \Longrightarrow z_i$ in P_i.

A *communication step* denoted by $(x_1, ..., x_n) \vdash (y_1, ..., y_n)$ is defined as follows:
Let, for $1 \leq i, j \leq n$,

$$\delta(x_i, j) = \begin{cases} \lambda, & \text{if } x_i \notin R_j \text{ or } i = j \\ x_i, & \text{if } x_i \in R_j \text{ and } i \neq j \end{cases}$$

Let, for $1 \leq j \leq n$, $\Delta(j) = \delta(x_1, j)\delta(x_2, j)...\delta(x_n, j)$
(this is the total message to be received by the j-th component)

And let, for $1 \leq i \leq n$, $\delta(i) = \delta(x_i, 1)\delta(x_i, 2)...\delta(x_i, n)$

(this is the total message sent by the i-th component).

Then for $1 \leq i \leq n$, we define

$$y_i = \begin{cases} \Delta(i), \text{ if } \Delta(i) \neq \lambda \\ x_i, \text{ if } \Delta(i) = \lambda \text{ and } \delta(i) = \lambda \\ S_i, \text{ if } \Delta(i) = \lambda \text{ and } \delta(i) \neq \lambda \end{cases}$$

Thus y_i can be:

- the concatenation of the messages received by the i-th component, if it receives at least one message, or
- the previous string, when the i-th component is not involved in communication, or
- S_i, if the i-th component sends messages but it does not receive message.

It should be observed that a component cannot send messages to itself.

Definition 6 *(Language) Given a CCPC Grammar system Γ the language generated by Γ is given by the following expression:*

$$L(\Gamma) = \left\{ \begin{array}{l} w \in T^* \mid (S_1, ..., S_n) \Longrightarrow (x_1^{(1)}, ..., x_n^{(1)}) \vdash (y_1^{(1)}, ..., y_n^{(1)}) \Longrightarrow \\ \Longrightarrow (x_1^{(2)}, ..., x_n^{(2)}) \vdash (y_1^{(2)}, ..., y_n^{(2)}) \Longrightarrow ... \Longrightarrow (x_1^{(s)}, ..., x_n^{(s)}), \\ for\ s \geq 1\ and\ w = x_1^{(s)} \end{array} \right\}$$

We denote by $CCPC_n(Y)$ the family of languages generated by Parallel Communicating Grammar systems with Communication by Command with at most n components, each component with productions of type Y, where:

- $n \in \mathbf{N} \cup \{*\}$ where symbol $*$ indicates that the parameter is irrelevant.
- $Y \in \{FIN, REG, CF, CS, RE\}$.

We introduce now Eco-Grammar systems, which can be seen as a generalization of both CD and PC Grammar systems.

2.3.3 Eco-Grammar systems

Eco-Grammar systems were introduced in [CVJKP94] as a formal framework for studying systems made up of a community of living organisms and their environment. They were inspired by eco-systems, economy, social behavior, multi-agent systems and collective robotic systems.

Definition 7 *(Eco-Grammar system) An EG system of degree $n \geq 1$ is a tuple $\Sigma = (E, A_1, \ldots, A_n)$, where:*

- *$E = (V_E, P_E)$, is the environment that uses V_E as a finite alphabet, and P_E as a finite set of 0L rewriting rules over V_E;*
- *A_i, $1 \leq i \leq n$, are agents consisting of $A_i = (V_i, P_i, R_i, \varphi_i, \psi_i)$ where:*
  - *V_i is a finite alphabet,*
  - *P_i a finite set of 0L rewriting rules over V_i,*
  - *$R_i \in V_E^+ \times V_E^*$ is a finite set of rewriting rules,*
  - *φ_i and ψ_i are computable functions, not necessarily complete, that respectively select production sets according to the environment state $\varphi_i : V_E^* \longrightarrow 2^{P_i}$, and according to the agent state $\psi_i : V_i^+ \longrightarrow 2^{R_i}$.*

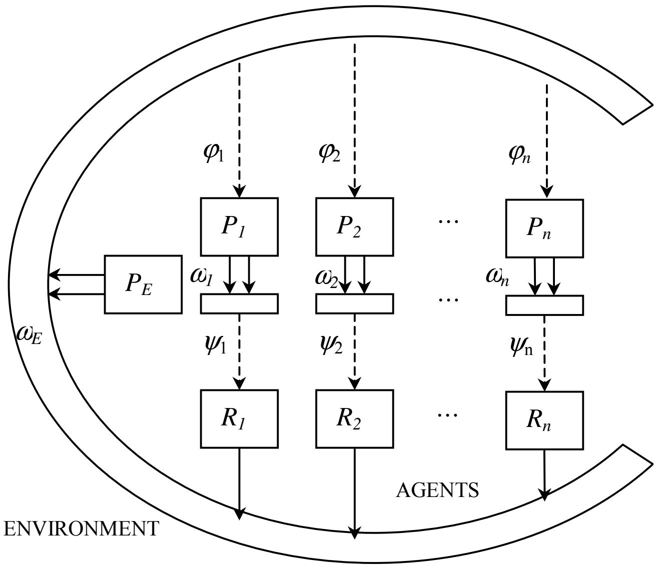

Figure 2.1: Description of an Eco-Grammar system

At this point, this definition provides only the description of the Eco-Grammar system's components. In order to describe the dynamic evolution of EGs, we give the definitions of configuration, derivation and language generated.

Definition 8 *(Configuration) A configuration of an Eco-Grammar system is a tuple $\sigma = (w_E, w_1, \ldots, w_n)$, $w_i \in V_i^*$, $i \in \{E\} \cup \{1, \ldots, n\}$, where w_E is a string that represents the environment state and $w_1, \ldots, w_n$ are strings that represent the agents' state.*

Below we define the direct derivation of a configuration in an Eco-Grammar system:

Definition 9 *(Derivation) Considering an Eco-Grammar system Σ and two configurations of Σ denoted by $\sigma = (w_E, w_1, \ldots, w_n)$ and $\sigma' = (w'_E, w'_1, \ldots, w'_n)$, we say that σ directly derives σ' written as $\sigma \Longrightarrow_\Sigma \sigma'$ iff*

- $w_i \Longrightarrow w'_i$ *according to the selected set of rules for the i-th agent by the* φ_i *mapping,*
- $w_E = z_1x_1z_2x_2, \ldots, z_mx_mz_{m+1}$ *and* $w'_E = z'_1y_1z'_2y_2, \ldots, z'_my_mz'_{m+1}$, *such that* $z_1x_1z_2x_2, \ldots, z_mx_mz_{m+1} \Longrightarrow z_1y_1z_2y_2, \ldots, z_my_mz_{m+1}$ *as the result of applying in parallel the rewriting rules selected by the* ψ_i *mappings, for all* $1 \leq i \leq n$, *and* $z_1z_2, \ldots, z_{m+1} \Longrightarrow z'_1z'_2, \ldots, z'_{m+1}$ *according to the environment's rules.*

The transitive and reflexive closure of $\Longrightarrow_\Sigma$ is denoted by $\Longrightarrow^+_\Sigma$ and $\Longrightarrow^*_\Sigma$ respectively.

Definition 10 *(Configuration sequences) For a given EG system* Σ *and an initial configuration* σ_0 *we define the set of configuration sequences of* Σ *as*

$$Seq(\Sigma, \sigma_0) = \{\{\sigma_i\}_{i=0}^{\infty} | \sigma_0 \Rightarrow_\Sigma \sigma_1 \Rightarrow_\Sigma ...\}$$

Definition 11 *(Language) The language generated by the environment is defined as*

$$L_E(\Sigma, \sigma_0) = \left\{ \begin{array}{c} w_E \in V_E^* \mid \sigma_j = (w_E, w_1, ..., w_n), \\ \sigma_0 \Longrightarrow_\Sigma \sigma_1 \Longrightarrow_\Sigma ... \Longrightarrow_\Sigma \sigma_j, j \geq 0 \end{array} \right\}$$

2.3.4 Reproductive Eco-Grammar system

Here we present a variant of EG system that allows to model agents' reproduction and birth, one characteristic feature of living organisms. This variant is known as Reproductive EG system and was introduced in [CVKKP97].

Definition 12 *(Reproductive EG system) A Reproductive EG system of degree n* ≥ 1 *is a (n+1)-tuple* $\Sigma = (E, \mathcal{A}_1, ..., \mathcal{A}_n)$ *where:*

- $E = (V_E, P_E)$ *is the environment, as defined for EG systems,*
- $\mathcal{A}_i$ *is a multiset (a set whose elements can occur in several copies each), called the population of the agents of the i-th type,* $1 \leq i \leq n$, *where each agent* A_i *in* $\mathcal{A}_i$ *has the same form* $A_i = (V_i \cup \{\sqcup\}, P_i, R_i, \varphi_i, \psi_i)$, *with the components* V_i, P_i, R_i, φ_i, ψ_i *defined as for EG systems and*
- $\sqcup$, *called the reproduction symbol, can occur only on the right-hand side of productions of* P_i, $1 \leq i \leq n$.

The appearance of the new agents is indicated by the occurrence of reproduction symbol $\sqcup$ in the current state of the agent. Whenever the current evolution stage w of an agent A has the form $w_1 \sqcup ... \sqcup w_n$, the agent is reproduced as a collection (a multiset) of agents $A^{(1)}, ..., A^{(n)}$ of the same type, where the agent $A^{(i)}$ is in evolution stage w_i, $1 \leq i \leq n$. The new agents $A^{(1)}, ..., A^{(n)}$ inherit all properties of their ancestor, namely they have the same sets of evolution rules and action rules.

The definition of configuration, derivation, configuration sequences and language for Reproductive EG systems is the same as for EG systems.

2.3.5 Conditional Tabled Eco-Grammar system

Conditional Tabled Eco-Grammar (CTEG) systems were introduced in [CVPS95] in the following way:

Definition 13 *A Conditional Tabled Eco-Grammar (CTEG) system of degree* $n \geq 1$ *is an EG system* $\Sigma = (E, A_1, ..., A_n)$ *where:*

- $E = (V_E, (e_1, f_1 : P_1), ..., (e_m, f_m : P_m))$ *is a conditional T0L scheme with n-ary context conditions,* $e_i, f_i \in V_1^* \times ... \times V_n^*$, $1 \leq i \leq m$ *and*
- $A_i = (V_i, (g_{i1}, h_{i1} : P_{i1}), ..., (g_{ir_i}, h_{i,r_i} : P_{i,r_i}))$, $1 \leq i \leq n$, *is a conditional T0L scheme with unary context conditions,* $g_{i1}, h_{i1} \in V_E^*$, *for all* i, j.

To specify the dynamic aspects of a CTEG system we need to introduce first the following definition:

Definition 14 *Given an alphabet* V*, we define the following predicates over* $V^* \times V^*$*:*

$$\begin{aligned} \pi_b(x,y) = 1 \quad & iff \quad y = y_1 x y_2, \\ \pi_s(x,y) = 1 \quad & iff \quad y = y_1 x_1 y_2 x_2 ... y_r x_r y_{r+1}, \\ & \qquad x = x_1 x_2 ... x_r \text{ where } x_i, y_i \in V^* \text{ for all } i. \end{aligned}$$

Whilst the definition of configuration and environment language for CTEG systems is the same as for EG systems, the notion of derivation is different:

Definition 15 *For a CTEG system* Σ, $c \in \{b, s\}$ *and configurations* $\sigma = (w_E, w_1, ..., w_n)$, $\sigma' = (w'_E, w'_1, ..., w'_n)$ *we write* $\sigma \Rightarrow_c \sigma'$ *iff:*

1. *There is a table* $(e_j, f_j : P_j)$ *in* E *such that* $\pi_c(e_{ji}, w_i) = 1$, $\pi_c(f_{ji}, w_i) = 0$, *for all* $1 \leq i \leq n$, *and* $w_E \Rightarrow_{P_j} w'_E$ *and*
2. *every* A_i, $1 \leq i \leq n$, *has a table* $(g_{ij}, h_{ij} : P_{ij})$ *for some* $1 \leq j \leq r_i$, *such that* $\pi_c(g_{ij}, w_E) = 1$, $\pi_c(h_{ij}, w_E) = 0$, *and* $w_i \Rightarrow_{P_{ij}} w'_i$.

We denote by $CTEG_n(i, j, c)$, $n \geq 1$, $c \in \{b, s\}$, $i, j \geq 0$ the family of languages $L_E(\Sigma, \sigma_0)$ where $\Sigma \in CTEG$ has degree n, initial configuration σ_0, all the permitting contexts have length at most i, all the forbidden contexts have length at most j and predicate π_c is used to verify the presence or absence of contexts. When the number of agents or the length of permitting or forbidding contexts is not bounded, then we replace the corresponding parameter with ∞.

When we are interested only in strings over some subalphabet T of V_E, hence in the language $L_E(\Sigma, \sigma_0) \cap T^*$, then we speak about *Extended Conditional Tabled Eco-Grammar (ECTEG) systems.* We denote by $ECTEG_n(i, j, c)$, $n \geq 1$, $c \in \{b, s\}$, $i, j \geq 0$ the family of languages $L_E(\Sigma, \sigma_0) \cap T^*$ where $\Sigma \in ECTEG$ has degree n, initial configuration σ_0, all the permitting contexts have length at most i, all the forbidden contexts have length at most j and predicate π_c is used to verify the presence or absence of contexts. When the number of agents, the length of permitting context or the length of the forbidding contexts is not bounded, we replace the corresponding parameter with ∞.

2.3.6 Some applications of Grammar System Theory

Much of the research in the area of Grammar systems is theoretical but there are practical applications such as: problem solvers [Mih95, SS04], simulators of real or artificial environments [Das95, CA00], dialogue models for the study of linguistic matters like: natural language processing, human dialogue simulations, cultural influences in linguistic phenomena [CVJL98, CVJLMV99, JL99, AJR01], etc. Even a special chip, the eco-chip, is proposed in [SM94] in order to physically realize an Eco-Grammar system.

With respect to the area of Artificial Intelligence, an Eco-Grammar system definition has been given in [Sos96] for modeling artificial neural networks. Similarly in [DG05b] we present an approach that relates evolutionary algorithms from Artificial Intelligence area with EG systems from Formal Language Theory. An *Evolutionary Algorithm* (EA) [B9̈6] is a computational model inspired by the Darwinian evolutionist theory. In nature, individuals coexisting in an environment respect a genetic theory of natural selection. The most adapted organisms have better chances to survive, to reproduce and to have offspring. Evolutionary algorithms maintain a population of structures that evolve according to the rules of recombination, mutation and selection. Although simplistic from a biologist's point of view, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms.

In [DG05b] we prove that starting from an arbitrary EA it is possible to construct an EG system that simulates the EA's behavior. For formal details refer to [DG05b]. Our motivation for connecting these two models is that on one hand theoretical basis for EAs can explain only partially the empirical results from numerous applications. And on the other hand the empirical use of EGs is very limited. By simulating EAs with EG systems we open the possibility to use these grammatical frameworks not just as language generators but also as searching problem solvers. The range of possible applications as problem solvers is broad, we mention here only several including the location-allocation problem (an NP-complete problem), game playing, face recognition, financial time-series prediction, etc. On one hand we believe that EAs can benefit by the theoretical results obtained in the framework of Grammar systems. On the other hand Grammar systems can take inspiration from applications using EAs for their theoretical research. Despite the large number of existing references in the area of EAs, it represents a research domain where theoretical proofs are still missing. Those computational models suggested by the Darwinian paradigm of evolution have been shown to be powerful and perform well on a broad class of problems. Yet, when the complexity of the applications increases, EAs exhibit some limitations, such as premature convergence. Convergence of an EA is defined as the process of multiplication of the same individual in the population. If the convergence process finds a *local optimum*, this is called *premature convergence*. Often, it is difficult for EAs to escape from such local optimum. Some approaches like variable control parameters [SP94] or parallel populations try to avoid the stagnation observed at the end of the evolution. In [DG05a] we continue with the approach presented in [DG05b] and we generalize it studying Eco-Grammar systems as models for parallel evolutionary algorithms.

In [CVJL98] and [JL99] a formal framework for studying cultural evolution is presented, using Eco-Grammar systems as a base. The introduced model is called *Cultural Eco-Grammar* system. Some new components are added to the original definition of EG systems and the relationships among the components are modified, in order to capture the characteristics of cultural systems.

The model can be seen as a multi-agent system where agents interact among themselves and with their interrelated environments (environment, cultures, subcultures...), playing an important role in cultural change. This formal model is introduced with the purpose of studying the evolution of natural languages, for explaining linguistic matters like syntax, pragmatics ("the study of the contribution of context to the understanding of language") and language changes.

In [JL00, JL01] the notion of Grammar system is extended to the notion of *Linguistic Grammar* system. A Linguistic Grammar system is a Grammar system whose components are not grammars, but Grammar systems as well. Therefore a Linguistic Grammar system can be seen as a macro-Grammar system composed of several micro-Grammar systems that are composed of several grammars. They show that Linguistic Grammar systems have all the features needed to be a formal model of natural language, namely: modularity, parallelism, interaction between modules, coordination, generation of non-context free structures with context free components, cooperation, distribution, etc.

In particular we are interested in the use of Grammar systems for modeling dialogues. In Chapter 4 we introduce some Grammar system variants for the specification of dialogue protocols and we present a brief summary of the state of the art in this field in the area of Grammar System Theory.

Chapter 3

A new approach for the verification and derivation of Grammar systems

3.1 Introduction

Although our aim in this thesis is to explore possible ways of applying Formal Language Theory to the formal study of dialogues, in this chapter we analyze an idea to contribute to Formal Language Theory, and in particular to Grammar System Theory, with theories from the Programming field. We address the problem of formally proving that a Grammar system verifies certain properties or that it produces a desired outcome using formal strategies of proof developed for concurrent programs.

In Formal Language Theory rewriting rules are a precise and unambiguous way to describe local changes. But getting an intuition of the global state of a grammar from its set of rewriting rules is undecidable and harder when the complexity of the grammar definition increases. In the case of Grammar systems the emergent, parallel and distributed behavior increases even more the computational complexity of the model and the level of difficulty in the proofs. Therefore the problem of formally proving properties in a Grammar system is a very important issue to take into consideration. Until now two formal strategies have been used for proving properties in a Grammar system:

- **Analysis by cases:** this technique consists in an exhaustive study of all the possible derivations in the system. Although being a formal proving strategy, most of the explanations of the considered cases are provided in natural language. We provide an example of its use with the proof of theorem 5 taken from [DP97].

- **Abstractions:** a particular kind of abstraction called *coverability tree* was first introduced in [TE93] for PC Grammar systems. In this tree the vertices correspond to vectors, where these vectors store the number of occurrences of each nonterminal in each sentential form of the system in a given configuration. The tree is made finite (and effectively constructible) by cutting off infinite paths, substituting them with leaves. An infinite path can be cut off, if the numbers of nonterminals in the sentential forms are not decreasing and no communications are involved. If the number of occurrences of a symbol is unboundedly increased in the path, it is denoted by a symbol w in the vector of the replacing leaf. Coverability trees allow to get decidable results about derivation steps and they were used for NPC Grammar systems with context free and context sensitive rules [TE93, TEIP94, TKI97] and for NPC Grammar systems with regular and linear production rules [Mih99b]. Coverability trees were also used for CD grammar systems in [Mih99a].

The Programming field is very developed in the area of formal derivation and verification of programs. *Verification* is used to prove that some program meets their specification and *formal derivation* is used when a program is developed hand in hand with the proof of satisfaction of its specification. A major drawback of verification is that it usually requires reconstructing the program that is being verified. And when the proposed program is not correct (or just cannot be shown to be) quite wasteful backtracking takes place. For this reason later efforts have concentrated on formal derivation, which has proved to be a much more economic way of constructing correct programs.

The imperative programming paradigm assumes that the computer can maintain through environments of variables any changes in a computation process. Computations are performed through a guided sequence of steps, in which these variables are referred to or changed. The order of the steps is crucial, because a given step will have different consequences depending on the current values of variables when the step is executed.

Inspired by the Programming field we want to contribute to Grammar System Theory with an alternative approach for the verification of properties and to introduce a novel strategy for performing formal derivation in Grammar systems.

We devote section 3.2 to explain how to perform the automatic translation of some Grammar systems to concurrent imperative programs because in the next sections we use this mechanism of translation to prove results. In section 3.2.1 we introduce the Dijkstra Guarded Command (DGC) language [DE76] that we use to define the multiprograms. In section 3.2.2 we explain with propositions 1, 2, 3 and 4 that CD Grammar systems, PC Grammar systems (centralized, non centralized, returning and non returning), CCPC Grammar systems and Eco-Grammar systems can be interpreted as imperative concurrent programs where each grammar is a program running concurrently. In the case of Reproductive Eco-Grammar systems, they can not be translated to the DGC language because this language does not provide instructions to dynamically create agent instances during run-time.

For those Grammar system variants that can be automatically translated to multiprograms we select the *Hoare logic* [Hoa69] as an axiomatic method for the formal verification and derivation of multiprograms. We chose Hoare logic because since its introduction in 1969 it has had a significant impact upon formal methods of formal verification and derivation of imperative sequential programs. Hoare's approach is perhaps the most well known verification formalism for imperative programming languages. In Hoare's work programs are seen as transformer of states or predicates. A program S is a finite sequence of statements. A statement denotes single commands (like assignments, conditional, loops, etc.). Statements take place one after another. That is, a statement does not begin until the preceding one has ended. To denote that a program S transforms a predicate P into a predicate Q the so-called Hoare triple $\{P\}S\{Q\}$ is used where P is the precondition of the program and Q is the postcondition. The precondition describes the set of initial states in which the program S is started and the postcondition describes the set of desirable final or output states. The analogy between programs as transformers of predicates and Grammar systems as derivation systems is clear: just as each statement in a program modifies the state of the system, every rewriting rule in the Grammar system modifies the string(s) under derivation. Informally a program is correct if it satisfies the intended input/output relation. To verify program correctness different Hoare-based verification systems have been defined, for instance [Rey82], [AL97], [dB99], [PdB03], [PHM99], [RWH01] and [vO01]. A verification system is defined as a set of verification rules, one rule for each type of statement in a programming language. The correctness of $\{P\}S\{Q\}$ is proved in the following way: using the verification rules a set of predicate logic formulas called verification conditions are generated. The proof of these verification conditions ensures the correctness of a program.

Owicki-Gries Theory[OG76] is the first complete, undecidable Hoare-based logic for proving partial correctness properties of concurrent programs with shared variables. Through this chapter we use this verification strategy for analyzing concurrent programs.

In section 3.3.1 we explain Owicki-Gries Theory. In section 3.3.2 we explain how Grammar systems can benefit from derivation, reasoning and proving strategies from the multiprogramming framework. For example: *given a Grammar system one can prove that it generates a specific language* by direct reasoning or one can translate the Grammar system into a multiprogram and prove the same statement by some programming strategies developed in the Owicki-Gries Theory. We exemplify this with the language $\{a^n b^n c^n \mid n \geq 0\}$. First with theorem 5 we introduce an example of proof using analysis by cases taken from [DP97]. After that we use the new strategy to present a different proof for the same theorem.

Furthermore, we propose another approach to solve problems of the following type: *given a language specification find a Grammar system that generates the given language*. The strategy widely used so far is as follows: first one proposes a Grammar system and then proves by means of language theory that the proposed Grammar system generates indeed the given language. With theorems 6, 7 and 8 we give three examples of how Owicki-Gries logic of programming could guide us in obtaining *simultaneously* a Grammar system that generates the given language and the proof that it generates it. This new approach might be of a great benefit for the Grammar System Theory. The strategy consists on translating the problem of finding a Grammar system Γ of certain type that generates a language L, into the problem of finding a multiprogram $\mathcal{P}$. $\mathcal{P}$ has as many programs $Prog_i$ as the number of grammars the Grammar system Γ has and $\mathcal{P}$ must be correct with respect to the specification:

$$\{(w_1 = S_1) \wedge (w_2 = S_2) \wedge ... \wedge (w_n = S_n) \wedge n \geq 1\}\, \mathcal{P}\, \{w_1 \in L\}.$$

Then this multiprogram is translated back into the Grammar system Γ, the whole behavior of Γ being similar to that of $\mathcal{P}$. Actually, the language generated by Γ is included in L, but for the examples we present here equality is reached, as detailed reasonings prove.

With theorem 6 we show how to apply this strategy to a well-known non-context free language, namely $L_{cd} = \{a^n b^m c^n d^m \mid n, m \geq 1\}$. In [Chi97] it was proved that L_{cd} can be generated by a returning non-centralized PC Grammar system with three context free components ($L_{cd} \in PC_3(CF)$). We improve this result showing that L_{cd} can be generated by a non-returning non-centralized PC Grammar system with five right regular components ($L_{cd} \in NPC_5(Reg)$). We provide thus a solution to an open problem mentioned in that work. This can be considered an improvement because though the number of grammars increases to five, its complexity is reduced from context free grammars to regular grammars. During the workshop "Grammar System Week" that took place in Budapest (Hungary) on July 2004 we presented this approach [GM04a]. With the help of Dr. Gheorghe Păun we found a better solution with respect to the number of right regular components of the Grammar system, proving that $L_{cd} \in NPC_3(Reg)$ based on a similar strategy. We give here the solution that we found in collaboration with Păun and with Theorem 9 we prove that it is the most economical one. Both results are included in the proceedings of the workshop [GM04b] and were selected by the workshop committee for later publication in [GM07].

Theorem 8 is an example of the use of simultaneous development and proof of multiprograms (Grammar systems) correctness, but also shows the combined use of Owicki-Gries strategy with some other programming techniques used for improving parallelism to obtain more time-efficient Grammar systems.

In section 3.3.3 we show how the concurrent Programming framework can benefit from Grammar System Theory to get negative results; in the first field there is no strategy to deal with negative results of the type: *a given language cannot be generated by any multiprogram (Grammar system) of a specified type*. This kind of problem has to be analyzed in the Grammar system framework, with the tools available there. We exemplify this with theorem 9 proving that $L_{cd} \notin X_2(Reg)$, for $X \in \{PC, CPC, NPC, NCPC\}$, i.e. that the cross-agreement language can not be generated by any type of PC Grammar system with two regular grammars.

In this chapter we analyze the formal verification and derivation of Grammar systems that can be translated to a multiprogramming in the DGC language. Reproductive Eco-Grammar systems can not be translated to the DGC language because this language does not provide instructions for the dynamic creation of agent instances that takes place in Reproductive Eco-Grammar systems. But Reproductive Eco-Grammar systems can be automatically translated to other programming paradigm that is called Object Oriented (OO) paradigm, which provides instructions to perform dynamic creation of objects (agent instances) during the execution of the program. We do not address this problem in this chapter but we conjecture that this is an interesting topic of research that should be investigated because some Hoare-based formal verification strategies are available for the OO paradigm, like for instance the ones explained in [Rey82], [AL97], [dB99], [PdB03], [PHM99], [RWH01] and [vO01].

3.2 Automatic translation of Grammar systems to concurrent imperative programs

In classic computing devices were centralized and computation was accomplished by one central processor. But in contemporary Computer Science distributed computing systems that consist on multiple communicating processors play a major role. The reason is illustrated by the advantages of this kind of system: efficiency, fault tolerance, scalability in the relation between price and performance, etc.

Since 1960, when the concept of *concurrent programming* [DE76] was introduced, a huge variety of topics related to parallelism and concurrency have been defined and investigated, such as operating systems, machine architectures, communication networks, circuit design, protocols for communication and synchronization, distributed algorithms, logics for concurrency, automatic verification and model checking. The same trend was observed in classic Formal Language and Automata Theory as well. In the beginning grammars and automata were modeling classic computing devices of one agent or processor, hence a language was generated by one grammar or recognized by one automaton. Inspired by different models of distributed systems in AI, *Grammar System Theory* [CVJJP94] has been developed as a grammatical theory for distributed and parallel computation. More recently, similar approaches have been reported for systems of automata [MVM00].

In the concurrent programming framework *Owicki-Gries Theory* [OG76], the first complete programming logic for formal development of concurrent programs, and another programming strategies were developed to help programmers in the analysis and derivation of multiprograms.

In order to provide Grammar systems with formal verification and derivation strategies from the programming field we need to have ways to translate them to programs in some programming language. We have chosen the Dijkstra's Guarded Command (DGC) language [DE76] for translating sequential programs running in parallel into a multiprogram. Our selection is justified by the fact that the Owicki-Gries Theory that we have selected to verify properties in multiprograms (Grammar systems) is defined in terms of the DGC language.

3.2.1 Dijkstra's Guarded Command language

The distinguishing features of DGC language are notational and mathematical austerity, and the incorporation of nondeterminacy in sequential programs. This language resembles ALGOL60, Pascal and C.

In DGC language elementary statements are:

- **skip**, which does nothing.
- **abort**, which is used to finish program execution abruptly.
- **Assignment** $x := E$. It evaluates expression E and then assigns its value to variable x.
- **Multiple assignment** $x, y := E, F$. It evaluates E and F simultaneously and then assigns their values to x and y respectively.

Out of existing statements one can construct new statements. They are:

- **Sequential composition** $S_0; S_1$. First S_0 is executed and then S_1.
- **Alternative construct**
```
 if  $B_0 \rightarrow S_0$
 |   ...
 |   $B_n \rightarrow S_n$
 fi
```
  The B's are called guards and the S's are the guarded statements. Any guard B_i that evaluates to *true* is nondeterministically selected, and the associated guarded statement S_i is performed. There are two possible interpretations for the case when no guard evaluates to *true*. In a sequential program it is interpreted as an *abort*. In a concurrent program the alternative construct where all the guards are false is equivalent to a busy wait, it keeps evaluating until one

of the B's becomes true. This second interpretation is inspired by the fact that when a program cooperates with other programs, it is possible that the activity of the other programs changes the false value of the guards to true, in which case continuation of the delayed alternative construct is possible.

- **Loop**

```
**do**  $B_0 \to S_0$
|       ...
|       $B_n \to S_n$
**od**
```

  This is just a loop, which keeps iterating until no guard B_i evaluates to *true*. During iteration it selects nondeterministically any guard B_i that evaluates to *true* and performs its associated guarded statement S_i. If no guard is *true* then the loop behaves like a skip command.

Below we formally introduce the semantics associated with each statement in the DGC language, which is expressed by Hoare triples:

- For **skip**:

$$\{P\}\ skip\ \{Q\} \leftrightarrow [P \Rightarrow Q]$$

- For **abort**:

$$\{P\}\ abort\ \{Q\} \leftrightarrow [P \leftrightarrow False]$$

- For **assignment**:

$$\{P\}\ x := E\ \{Q\} \leftrightarrow [P \Rightarrow Q^x_E]$$

  where Q^x_E is the predicate resulting by replacing all the occurrences of variable x for expression E.

- For **multiple assignment**:

$$\{P\}\ x, y := E, F\ \{Q\} \leftrightarrow [P \Rightarrow Q^{x,y}_{E,F}]$$

  where $Q^{x,y}_{E,F}$ is the predicate resulting from the simultaneous replacement of all the occurrences of variables x and y for expressions E and F, respectively.

- For **sequential composition**:

$$\{P\}\ S_0; S_1\ \{R\} \leftrightarrow \exists\ Q \text{ such that } \{P\}\ S_0\ \{Q\} \wedge \{Q\}\ S_1\ \{R\}$$

- For **alternative construct**:

$$\begin{array}{lllcl} \{P\} & \textbf{if} & B_0 \to S_0 \quad \{Q\} & \leftrightarrow & P \Rightarrow [B_0 \vee \ldots \vee B_n] \\ & | & \ldots & & \wedge\ \{P \wedge B_0\}\ S_0\ \{Q\} \\ & | & B_n \to S_n & & \ldots \\ & \textbf{fi} & & & \wedge\ \{P \wedge B_n\} S_n\ \{Q\} \end{array}$$

In case all the guards are false the guarded statements $S_0, ..., S_n$ may fail to establish the desired precondition Q. So no guarded command can be selected for execution and the alternative construct becomes stuck. But according to the definition of correct Hoare triple the execution of the alternative construct can either terminate in a state satisfying Q or not terminate at all. So a very viable implementation of the alternative construct when all the guards are false is a busy wait.

- For **loop**:
  We use the so-called *invariance theorem for repetitive constructs* which was formulated by Hoare in [Hoa69]. This theorem says that for an *invariant of the repetition P* the following condition must be satisfied:

$$\begin{array}{llll} \{P\} & \textbf{do} & B_0 \to S_0 \quad \{Q\} & \leftrightarrow \quad [P \wedge \neg B_0 \wedge ... \wedge \neg B_n \Rightarrow Q] \\ & | & ... & \qquad \wedge \{P \wedge B_0\}\ S_0\{P\} \\ & | & B_n \to S_n & \qquad ... \\ & \textbf{od} & & \qquad \wedge\{P \wedge B_n\}\ S_n\ \{P\} \end{array}$$

In [DS90] another definition for loop is given, based on the definitions of weakest liberal precondition (*wlp*) and fixpoint equation.

3.2.2 How to translate Grammar systems to multiprograms

Before explaining how to automatically translate some Grammar system variants to equivalent concurrent programs written in DGC language, we introduce below the formal definition of equivalence that we use:

Definition 16 *A Grammar system* $\Gamma \in \{CD, PC, CPC, NPC, NCPC\}$ *is equivalent to a multiprogram* $\mathcal{P}$ *iff for every string* $w_j \in L(\Gamma)$ *there is a computation in* $\mathcal{P}$ *which generates the string* w_j *as output, and for every computation in* $\mathcal{P}$ *that generates as output the string* w_k *it happens that* $w_k \in L(\Gamma)$.

Now we can introduce the corresponding results:

Proposition 1 *For every CD Grammar system* $\Gamma = (T, G_1, G_2, .., G_n, S)$ *a concurrent program that is equivalent to* Γ *can be constructed.*

Proof. From an arbitrary CD Grammar system Γ of degree $n \geq 1$ we can define the multiprogram $Prog_\Gamma$ from figure 3.1 that is equivalent to Γ. In $Prog_\Gamma$ a global string w is introduced to represent the sentential form that in Γ grammars $G_1, G_2, .., G_n$ collaborate to derive. In Γ the sentential form w is arbitrarily assigned in turns to the grammar components. After a grammar finishes performing derivations over w the turn is nondeterministically assigned to other grammar in Γ. In $Prog_\Gamma$ an integer variable *grammar* is introduced to denote the grammar component that is nondeterministically selected to derive string w. Function $Value$ is used to arbitrarily select a number between 1 and n. While the string w contains non terminals the programs $Prog_1, ..., Prog_n$ corresponding to grammars $G_1, ..., G_n$ iterate. $Prog_\Gamma$ finishes when w is a string of terminal symbols from set T. In this way every computation in $Prog_\Gamma$ that generates as output a string w halts in a state satisfying $\{w \in L(\Gamma)\}$.

Each program $Prog_i$ in the multiprogram $Prog_\Gamma$ corresponds to a grammar G_i in the CD Grammar system Γ. Depending on the derivation mode f_i of grammar G_i a different program $Prog_i$ is defined. In figures 3.2 to 3.6 we introduce the corresponding programs $Prog_i$ for grammars G_i with derivation modes t, $= k$, $\leq k$, $\geq k$ and $*$ respectively. Independently of the derivation mode all the grammars G_i have to wait for being assigned the sentential form w before executing rewriting rules. Therefore all the programs $Prog_i$ are defined as an infinite loop with an alternative command with condition $grammar = i$ that guarantes that the program keeps waiting until that condition is satisfied. When $grammar = i$ program $Prog_i$ is assigned the processor until it finishes rewriting string w in the same way as grammar G_i would derive sentential form w. In $Prog_i$ the sequence of commands that simulates the way grammar G_i derives sentential form w is embraced by symbols $\langle$ and $\rangle$. In Owicki-Gries Theory the commands embraced by symbols $\langle$ and $\rangle$ are considered atomic, during its execution the processor can not be assigned to other active processes. By the alternative construction *if* programs $Prog_i$ presented in figures 3.2 to 3.6, nondeterministically choose to rewrite string w as rewriting rules P_i are nondeterministically chosen by grammars G_i to rewrite the sentential form w under derivation. Therefore for every string $w \in L(\Gamma)$ there is a computation in $Prog_\Gamma$ which generates as output w and halts in a state satisfying $\{w \in L(\Gamma)\}$.

The program corresponding to a Grammar system G_i with terminal derivation mode is introduced in figure 3.2. The loop with condition $flag$ simulates the t-mode derivation performed by the grammar G_i. In the alternative command there is one guarded command for each rule in P_i plus one extra guarded command. Each guarded command, except the last one, has the shape $Contains(w, \alpha) \rightarrow Rewrite(\alpha, \beta, w)$ where $\alpha \rightarrow \beta$ is a rewriting rule from P_i. The last guarded command does not correspond to a rewriting rule in P_i because it falsifies the boolean variable $flag$ when no rewriting rule from P_i can be applied over sentential form w. When $flag$ is false the program exits the loop and the variable *grammar* is actualized with a new arbitrary value between 1 and n, corresponding to the next grammar that can rewrite string w.

The program corresponding to a Grammar system G_i with $= k$ derivation mode is introduced in figure 3.3. An integer variable *cont* is used to count the number of times the program rewrites string w. Initially $cont = 0$. The loop with condition $cont \neq k$ simulates the derivation performed by grammar G_i in $= k$-mode. In the alternative command there is one guarded command for each rule in P_i. In the loop the alternative command is followed by an increment of variable *cont*. When $cont = k$ the program exits the loop after k rewriting rules have been applied over string w. After the execution of the loop, the variable *grammar* is actualized with a new arbitrary value between 1 and n, corresponding to the next grammar that can rewrite w.

The program corresponding to a Grammar system G_i with $\leq k$ derivation mode is introduced in figure 3.5. A variable *cont*, initialized with value zero, is used to count the number of times the program rewrites string w. The loop with condition $cont \neq k$ simulates the derivation performed by grammar G_i in $\leq k$-mode. In the alternative command there is one guarded command for each rule in P_i plus one extra guarded command. The last guarded command can be arbitrarily selected to simulate the execution of no rewriting rule. In the loop the alternative command is followed by an increment of variable *cont*. When $cont = k$ the program exists the loop after $\leq k$ rewriting rules have been applied over string w. After exiting the loop the variable *grammar* is actualized with a new arbitrary value between 1 and n, corresponding to the next grammar that can rewrite w.

The program corresponding to a Grammar system G_i with $\geq k$ derivation mode is introduced in figure 3.5. A variable *cont*, initialized to zero, is used to count the number of times the program rewrites string w. The loop with condition $cont \neq k$ simulates the derivation performed by grammar G_i in $\leq k$-mode. In the alternative command there is one guarded command for each rule in P_i plus one extra guarded command. The last guarded command can be arbitrarily selected to simulate the execution of no rewriting rule. In the loop the alternative command is followed by an increment of variable *cont*. When $cont = k$ the program exists the loop after k rewriting rules have been applied over string w. After exiting the first loop a boolean variable $flag$ is initialized with value *true*. The second loop with condition $flag$ simulates the derivation performed by grammar G_i in $*$-mode. In the alternative command there is one guarded command for each rule in P_i plus one extra guarded command. The last guarded command can be arbitrarily chosen to falsify the variable $flag$. It can be selected when no rewriting rule from P_i can be used or when there are rewriting rules to apply but none of them is selected. The alternative command in the second loop is followed by an increment of variable *cont*. When the variable $flag$ is $false$ the program exits the second loop and finishes. When the program finishes $\geq k$ rewriting rules have been applied over string w. After exiting the loop the variable *grammar* is actualized with a new arbitrary value between 1 and n, corresponding to the next grammar that can rewrite w.

The program corresponding to a Grammar system G_i with $*$ derivation mode is introduced in figure 3.6. The behavior of this program is identical to the second loop in the program from figure 3.5. ■

```
$Prog_\Gamma$ :  $n : Constant;$
         $w : String;$
         $grammar : Integer;$
         $N_1, ..., N_n : Set\ of\ Character;$

         $N_1, ..., N_n, w := N_{1,0}, ..., N_{n,0}, S;$
         $\{w = S\}$
         $grammar := Value(1, n);$
         $do\ Contains(w, N_1 \cup ... \cup N_n) \rightarrow$
               $Prog_1 \parallel Prog_2 \parallel ... \parallel Prog_n;$
         $od;$
         $\{w \in L(\Gamma)\}$
         $Print(w)$
```

Figure 3.1: Multiprogram for a CD Grammar system

```
$Prog_i$ :  $P_i : Set\ of\ Rule;$
            $flag : Boolean;$
            $do\ true \rightarrow$
                $if\ grammar = i \rightarrow$   $\langle \{w = \alpha \wedge grammar = i\}$
                                    $flag := true;$
                                      $do\ flag \rightarrow$
                                        $\left\{ Inv : w = \beta \wedge \alpha \overset{*}{\underset{P_i}{\Longrightarrow}} \beta \right\}$
                                            $if$     $...$
                                                   $| ...$
                                                   $| \neg CanApply(w, P_i) \rightarrow flag := false;$
                                            $fi;$
                                      $od;$
                                    $\left\{ w = \beta \wedge \alpha \overset{*}{\underset{P_i}{\Longrightarrow}} \beta \wedge (\nexists \delta : \beta \overset{*}{\underset{P_i}{\Longrightarrow}} \delta) \wedge grammar = i \right\}$
                                     $grammar := Value(1, n); \rangle$
                $fi;$
            $\{w = \beta \wedge \alpha \overset{t}{\underset{P_i}{\Longrightarrow}} \beta \wedge grammar = j\}$
            $od;$
```

Figure 3.2: Program for a CD grammar component with terminal mode of derivation

```
$Prog_i$ :  $k, cont : Integer;$
            $P_i : Set\ of\ Rule;$
            $\{w_i = \alpha\}$
            $k := value;$
            $do\ true \rightarrow$
                $if\ grammar = i \rightarrow$   $\langle \{w = \alpha \wedge grammar = i \wedge k = value\}$
                                     $cont := 0;$
                                      $do\ cont \neq k \rightarrow$
                                        $\left\{ Inv : w = \beta \wedge \alpha \overset{cont}{\underset{P_i}{\Longrightarrow}} \beta \wedge cont < k \wedge k = value \right\}$
                                                $if$  $...$
                                                    $| ...$
                                                $fi;$
                                                $cont := cont + 1;$
                                      $od;$
                                    $\left\{ w = \beta \wedge \alpha \overset{cont}{\underset{P_i}{\Longrightarrow}} \beta \wedge cont = k \wedge grammar = i \wedge k = value \right\}$
                                      $grammar := Value(1, n); \rangle$
                $fi;$
            $\{w = \beta \wedge \alpha \overset{=k}{\underset{P_i}{\Longrightarrow}} \beta \wedge grammar = j \wedge k = value\}$
            $od;$
```

Figure 3.3: Program for a CD grammar component with $= k$ mode of derivation

```
$Prog_i :$  $k, cont : Integer;$

        $\{w = \alpha\}$
        $k := value;$
        $do\ true \rightarrow$
                $if\ grammar = i \rightarrow$   $\langle\ \{w = \alpha \wedge grammar = i \wedge k = value\}$
                                        $cont := 0;$
                                        $do\ cont \neq k \rightarrow$
                                        $\left\{\begin{array}{l} Inv : w = \beta \wedge \alpha \overset{s}{\underset{P_i}{\Longrightarrow}} \beta \wedge \\ s \leq cont \wedge cont < k \wedge k = value \end{array}\right\}$
                                                    $if$    ...
                                                        | ...
                                                        $|\ true \rightarrow skip;$
                                                    $fi;$
                                                    $cont := cont + 1;$
                                        $od;$
                                        $\left\{\begin{array}{l} w = \beta \wedge \alpha \overset{s}{\underset{P_i}{\Longrightarrow}} \beta \wedge s \leq cont \wedge cont = k \wedge \\ grammar = i \wedge k = value \end{array}\right\}$
                                        $grammar := Value(1, n);\ \rangle$
                $fi;$
        $\{w = \beta \wedge \alpha \overset{\leq k}{\underset{P_i}{\Longrightarrow}} \beta \wedge grammar = j \wedge k = value\}$
        $od;$
```

Figure 3.4: Program for a CD grammar component with $\leq k$ mode of derivation

```
$Prog_i$ :  $k : Integer;$
            $flag : Boolean;$

            $\{w = \alpha\}$
            $k := value;$
            $do\ true \rightarrow$
                    $if\ grammar = i \rightarrow$   $\langle\ \{w = \alpha \wedge grammar = i \wedge k = value\}$
                            $cont := 0;$
                            $do\ cont \neq k \rightarrow$
                            $\left\{\ Inv : w = \beta \wedge \alpha \underset{P_i}{\overset{cont}{\Longrightarrow}} \beta \wedge cont < k \wedge k = value\ \right\}$
                                    $if$  ...
                                        | ...
                                        $|\ true \rightarrow skip;$
                                    $fi;$
                                    $cont := cont + 1;$
                            $od;$
                            $\left\{\ w = \beta \wedge \alpha \underset{P_i}{\overset{=k}{\Longrightarrow}} \beta \wedge grammar = i \wedge k = value\ \right\}$
                            $flag := true;$
                            $do\ flag \rightarrow$
                              $\left\{\ Inv : w = \delta \wedge \beta \underset{P_i}{\overset{*}{\Longrightarrow}} \delta \wedge flag \wedge k = value\ \right\}$
                                    $if$  ...
                                        | ...
                                        $|\ true \rightarrow flag := false;$
                                    $fi;$
                                    $cont := cont + 1;$
                            $od;$
                            $\left\{\begin{array}{l} w = \delta \wedge \alpha \underset{P_i}{\overset{=k}{\Longrightarrow}} \beta \underset{P_i}{\overset{*}{\Longrightarrow}} \delta \wedge \neg flag \wedge \\ grammar = i \wedge k = value \end{array}\right\}$
                            $grammar := Value(1, n);\ \rangle$
                    $fi;$
            $\{w = \beta \wedge \alpha \underset{P_i}{\overset{\geq k}{\Longrightarrow}} \beta \wedge grammar = j \wedge k = value\}$
            $od;$
```

Figure 3.5: Program for a CD grammar component with $\geq k$ mode of derivation

```
$Prog_i$ :  $P_i : Set\ of\ Rule;$
            $Flag : Boolean;$

            $\{w = \alpha\}$
            $do\ true \rightarrow$
                    $if\ grammar = i \rightarrow$  $\langle\ \{w = \alpha \wedge grammar = i\}$
                                          $flag := true;$
                                          $do\ flag \rightarrow$
                                            $\left\{\ Inv : w = \beta \wedge \alpha \overset{*}{\underset{P_i}{\Longrightarrow}} \beta \wedge flag\ \right\}$
                                                  $if$     ...
                                                         | ...
                                                         | $true \rightarrow flag := false;$
                                                  $fi;$
                                          $od;$
                                          $\left\{\ w = \beta \wedge \alpha \overset{*}{\underset{P_i}{\Longrightarrow}} \beta \wedge \neg flag\ \right\}$
                                          $grammar := Value(1, n);\ \rangle$
                    $fi;$
            $\{w = \beta \wedge \alpha \overset{*}{\underset{P_i}{\Longrightarrow}} \beta \wedge grammar = j\}$
            $od;$
```

Figure 3.6: Program for a CD grammar component with $*$ mode of derivation

Proposition 2 *For every PC, CPC, NPC and NCPC Grammar system* $\Gamma = (N, K, T, (P_1, S_1), ..., (P_n, S_n))$ *a concurrent program that is equivalent to* Γ *can be constructed.*

Proof.

From an arbitrary PC Grammar system Γ of degree $n \geq 1$ we can construct the multiprogram $Prog_\Gamma$ depicted in figure 3.7 that is equivalent to Γ. For each grammar component (P_i, S_i) in Γ a program $Prog_i$ like the one in figure 3.9 is defined in multiprogram $Prog_\Gamma$. In $Prog_\Gamma$ n strings $w_1, ..., w_n$ are introduced, one string w_i per each grammar component. If it is a centralized PC Grammar system then $Condition = Contains(w_1, N)$. While the string w_1 contains non terminals the programs $Prog_1, ..., Prog_n$ corresponding to the n grammar components iterate. $Prog_\Gamma$ finishes when w_1 is a string of terminal symbols from set T. On the other hand if Γ is a non centralized PC Grammar system then $Condition = Contains(w_1, N) \vee ... \vee Contains(w_n, N)$. While strings $w_1, ..., w_n$ contain non terminals the programs $Prog_1, ..., Prog_n$ iterate. The program $Prog_\Gamma$ finishes when some string contains only terminal symbols from set T. In $Prog_\Gamma$ the concurrent execution of programs $Prog_1, ..., Prog_n$ is followed by an alternative command with different conditions $contains(w_1, K) \vee ... \vee contains(w_n, K)$ and $\neg(contains(w_1, K) \vee ... \vee contains(w_n, K))$. In this way what in Γ is a derivation step followed by a possible communication step is simulated in $Prog_\Gamma$. In case the condition $contains(w_1, K) \vee ... \vee contains(w_n, K)$ is true, a loop that simulates the communication step is performed. The definition of this loop depends on Γ being a non returning or a returning PC Grammar system.

Each execution of program $Prog_i$ simulates one derivation step over string w_i according to rules P_i. Therefore each program $Prog_i$ is defined as an alternative command. In the alternative command there is one guarded command for each rule in P_i plus one extra guarded command. The last guarded command is used when no rewriting rule can be applied. Every computation in $Prog_\Gamma$ that generates an output w halts in a state satisfying $\{w_1 \in L(\Gamma)\}$. By the alternative construction programs $Prog_i$ nondeterministically choose to rewrite string w_i as grammars G_i nondeterministically choose to rewrite the sentential form w_i using rules P_i. Therefore for every string $w_1 \in L(\Gamma)$ there is a computation in $Prog_\Gamma$ that generates an output w_1 and halts in a state satisfying $\{w_1 \in L(\Gamma)\}$. ■

```
$Prog_\Gamma$ :  $n : Constant;$
         $N, K, T : Set\ of\ String;$
         $w_1, ..., w_n : String;$

         $N, K, T, w_1, ..., w_n := N_0, K_0, T_0, S_1, ..., S_n;$
         $\{(w_1, ..., w_n) = (S_1, ..., S_n)\}$
         $do\ Condition \rightarrow$
                  $\{(w_1, ..., w_n) = (\alpha_1, ..., \alpha_n)\}$
                  $Prog_1 \parallel Prog_2 \parallel ... \parallel Prog_n;$
                  $\{(w_1, ..., w_n) = (\beta_1, ..., \beta_n) \wedge (\alpha_1, ..., \alpha_n) \underset{\Gamma}{\Longrightarrow} (\beta_1, ..., \beta_n)\}$
                  $if\quad contains(w_1, K) \vee ... \vee contains(w_n, K) \rightarrow\ communicate(w_1, ..., w_n);$
                     $|\ \neg(contains(w_1, K) \vee ... \vee contains(w_n, K)) \rightarrow\ skip;$
                  $fi;$
                  $\left\{ \begin{array}{l} ((w_1, ..., w_n) = (\delta_1, ..., \delta_n) \wedge (\beta_1, ..., \beta_n) \vdash (\delta_1, ..., \delta_n)) \vee \\ \left( \begin{array}{l} (w_1, ..., w_n) = (\beta_1, ..., \beta_n) \wedge \\ \neg(contains(w_1, K) \vee ... \vee contains(w_n, K)) \end{array} \right) \end{array} \right\}$
         $od;$
         $\{w_1 \in L(\Gamma)\}$
         $Print(w_1)$
```

Figure 3.7: Multiprogram for a PC Grammar system

```
$Prog_i$ :  $P_i : Set\ of\ Rule;$

      $\{w_i = \alpha\}$
       $if\quad ...$
          $|\ ...$
          $|\ \neg CanApply(w_i, P_i) \rightarrow skip;$
       $fi;$
      $\{(w_i = \beta \wedge \alpha \underset{P_i}{\Longrightarrow} \beta) \vee (w_i = \alpha \wedge (\nexists \beta : \alpha \underset{P_i}{\Longrightarrow} \beta))\}$
```

Figure 3.8: Program for a PC grammar component

Proposition 3 *For every CCPC Grammar system* $\Gamma = (N, T, (S_1, P_1, R_1), ..., (S_n, P_n, R_n))$ *a concurrent program that is equivalent to* Γ *can be constructed.*

Proof.

From an arbitrary CCPC Grammar system Γ of degree $n \geq 1$ we can construct the multiprogram $Prog_\Gamma$ from figure 3.10 that is equivalent to Γ. For each grammar component (S_i, P_i, R_i) in Γ a program $Prog_i$ like the one from figure 3.9 is defined in multiprogram $Prog_\Gamma$. In $Prog_\Gamma$ n strings $w_1, ..., w_n$ are introduced, one string w_i per each sentential form in Γ. While the string w_1 contains non terminals the programs $Prog_1, ..., Prog_n$ iterate. The program $Prog_\Gamma$ finishes when w_1 is a string of terminal symbols from set T. In Γ the grammar components contribute to the derivation process performing rewriting steps followed by communication steps. In $Prog_\Gamma$ when the concurrent execution of programs $Prog_1, ..., Prog_n$ has finished a rewriting step in Γ has been simulated. After that the last loop in $Prog_\Gamma$ simulates a communication step in Γ.

In program $Prog_i$ each execution of the loop with condition $flag$ simulates over strings w_i the rewriting that component $G_i = (S_i, P_i, R_i)$ can perform over the sentential form w_i in a rewriting step in Γ. With the execution of the loop the program $Prog_i$ rewrites the string w_i an arbitrary number of times, simulating the $*$-derivation mode of component G_i. In the alternative command that defines the body of the loop for each rewriting rule in P_i an alternative command is defined, except the last guarded command. The last guarded command is used to arbitrarily exit the loop. Therefore every computation in $Prog_\Gamma$ that generates an output w_1 halts in a state satisfying $\{w_1 \in L(\Gamma)\}$. By the alternative construction *if* programs $Prog_i$ nondeterministically choose to rewrite string w_i as grammars G_i nondeterministically choose to rewrite the sentential form w_i using rewriting rules P_i. Then for every string $w_1 \in L(\Gamma)$ there is a computation in $Prog_\Gamma$ which generates the string w_1 and halts in a state satisfying $\{w_1 \in L(\Gamma)\}$. ■

```
$Prog_i$ :   $flag : Boolean$;

        ⟨  $\{w_i = \alpha\}$
           $flag := true$;
           $do\ flag \rightarrow$
               $\{Inv : w_i = \beta \wedge \alpha \overset{*}{\underset{P_i}{\Longrightarrow}} \beta \wedge flag\}$
                         $if$    ...
                                 | ...
                                 | $true \rightarrow flag := false$;
                         $fi$;
           $od$;
           $\{w_i = \beta \wedge \alpha \overset{*}{\underset{P_i}{\Longrightarrow}} \beta \wedge \neg flag\}$ ⟩
```

Figure 3.9: Program for a CCPC grammar component

```
$Prog_\Gamma$ :  $n : Constant;$
         $w_1, ..., w_n : String;$
         $T, N : Set\ of\ Character;$
         $i : Integer;$
         $R_1, ..., R_n : String;$

         $T, N, R_1, ..., R_n, w_1, ..., w_n := T_0, N_0, R_{1,0}, ..., R_{n,0}, S_1, ..., S_n;$
         $\{(w_1, ..., w_n) = (S_1, ..., S_n)\}$
         $do\ contains(w_1, N) \rightarrow$
            $\{(w_1, ..., w_n) = (\alpha_1, ..., \alpha_n)\}$
               $q := 0;$
               $Prog_1 \parallel Prog_2 \parallel ... \parallel Prog_n;$
            $\{(w_1, ..., w_n) = (\beta_1, ..., \beta_n) \wedge (\alpha_1, ..., \alpha_n) \Rightarrow (\beta_1, ..., \beta_n)\}$
               $i := 1;$
               $do\ i \leq n \rightarrow$
                     $if$          $\Delta(i) \neq \lambda \rightarrow w_i := \Delta(i);$
                                 $| \Delta(i) = \lambda \wedge \delta_(i) = \lambda \rightarrow skip;$
                                 $| \Delta(i) = \lambda \wedge \delta(i) \neq \lambda \rightarrow w_i := S_i;$
                     $fi;$
                     $i := i + 1;$
               $od;$
            $\{(w_1, ..., w_n) = (\pi_1, ..., \pi_n) \wedge (\beta_1, ..., \beta_n) \vdash (\pi_1, ..., \pi_n)\}$
         $od;$
         $\{w_1 \in L(\Gamma)\}$
         $Print(w_1)$
```

Figure 3.10: Multiprogram for a CCPC Grammar system

Definition 17 *An Eco-Grammar system $\Gamma = (E, A_1, \dots, A_n)$ is equivalent to a multiprogram P iff the set $L(\Gamma)$ is equal to the set of outputs generated by multiprogram $\mathcal{P}$.*

Proposition 4 *For every Eco-Grammar system $\Gamma = (E, A_1, \dots, A_n)$ with initial configuration $\sigma_0 = (w_{E,0}, w_{1,0}, ..., w_{n,0})$ a concurrent program that is equivalent to Γ can be constructed.*

Proof. From an arbitrary Eco-Grammar system Γ of degree $n \geq 1$ we can construct the multiprogram $Prog_\Gamma$ in figure 3.11 that is equivalent to Γ. In $Prog_\Gamma$ $n+1$ strings $w_E, w_1, ..., w_n$ are defined. Each string w_i represents the state of agent A_i in Γ. The string w_E represents the environment state in Γ. The strings $w_E, w_1, ..., w_n$ are initialized with the values $w_{E,0}, w_{1,0}, ..., w_{n,0}$ from initial configuration σ_0. According to definition 9 derivations in Eco-Grammar systems are potentially infinite. Therefore multiprogram $Prog_\Gamma$ is defined as the infinite execution of the sequential composition of concurrent programs $Prog_{\varphi_1}, Prog_{\psi_1}, ..., Prog_{\varphi_n}, Prog_{\psi_n}$ with program $Prog_{P_E}$. Each pair of programs $Prog_{\varphi_i}$ and $Prog_{\psi_i}$ in $Prog_\Gamma$ correspond to the execution in Γ of mappings φ_i and ψ_i in agent A_i. And program $Prog_{P_E}$ corresponds to the execution in Γ of environment rules P_E. Therefore the set of strings w_E printed by program $Prog_\Sigma$ is included in the set $L(\Sigma)$.

Program $Prog_{P_E}$ introduced in figure 3.12 corresponds to the execution of the environment evolution rules P_E over string w_E. For each rule r from set P_E a loop with condition $\delta = \alpha x \beta \wedge \neg Contain(\alpha, x)$ is performed to simulate the parallel rewriting of rule r in w_E.

Programs $Prog_{\varphi_i}$ and $Prog_{\psi_i}$ are introduced in figures 3.13 and 3.14 to simulate the execution in Σ of mappings φ_i and ψ_i respectively. With each rule r selected from sets P_i and R_i according to the content of strings $oldw_E$ and $oldw_i$, the programs respectively rewrite strings w_i and w_E. It is not mandatory for the computer where program $Prog_\Sigma$ is executed to assign one processor to each program $Prog_{\varphi_1}, Prog_{\psi_1}, ..., Prog_{\varphi_n}, Prog_{\psi_n}$. In case the computer has less than $n * 2$ processors they will be assigned in turns to the different programs to simulate a concurrent execution. Then it is possible for a program $Prog_{\psi_j}$ to modify string w_E before program $Prog_{\varphi_j}$ selects rules according to the environment state or it is possible for a program $Prog_{\varphi_j}$ to modify string w_i before program $Prog_{\psi_j}$ selects rules according to the agent state. In order to avoid these problems auxiliary variables $oldw_E, oldw_1, ..., oldw_n$ are introduced and used to select rules from sets P_i and R_i respectively.

The inclusion of the set $L(\Sigma)$ into the set of outputs generated by program $Prog_\Sigma$ is guarranted because function $Take$ used in the programs $Prog_{P_E}$, $Prog_{\varphi_i}$ and $Prog_{\psi_i}$ arbitrarily selects a rule from a set of rewriting rules selected by evolution rules P_E or by mappings φ_i and ψ_i, to be applied over strings w_E and w_i respectively. ■

```
$Prog_\Gamma$ :  $n : Constant;$
         $w_E, w_1, ..., w_n, oldw_E, oldw_1, ..., oldw_n : String;$

         $w_E, w_1, ..., w_n := w_{E,0}, w_{1,0}, ...w_{n,0};$
         $\{(w_E, w_1, ..., w_n) = (w_{E,0}, w_{1,0}, ...w_{n,0})\}$
         $do\ true \rightarrow$
                  $\{(w_E, w_1, ..., w_n) = (\delta, \alpha_1, ..., \alpha_n)\}$
                  $oldw_E, oldw_1, ..., oldw_n := w_E, w_1, ..., w_n;$
                  $Prog_{\varphi_1} \parallel Prog_{\psi_1} \parallel ... \parallel Prog_{\varphi_n} \parallel Prog_{\psi_n};$
                  $\left\{ \begin{array}{l} (w_E, w_1, ..., w_n) = (\pi, \beta_1, ..., \beta_n) \wedge \\ \delta \underset{\psi_1 \times ... \times \psi_n}{\Longrightarrow} \pi \wedge \left( \forall 1 \leq i \leq n : \delta_i \underset{\varphi_i}{\Longrightarrow} \beta_i \right) \end{array} \right\}$
                  $Prog_{P_E};$
                  $\left\{ (w_E, w_1, ..., w_n) = (\rho, \beta_1, ..., \beta_n) \wedge \pi \underset{P_E}{\Longrightarrow} \rho \wedge w_E \in L(\Gamma) \right\}$
                  $Print(w_E);$
         $od;$
```

Figure 3.11: Multiprogram for an Eco-Grammar system

```
$Prog_{P_E}$ :  $R, P_E : Set\ of\ Rule;$
                $\delta, \alpha, \beta, \varepsilon : String;$
                $r : Rule;$
                $x : Character;$

                $\{w_E = \alpha\}$
                $w_E = 0L\_rewrite(w_E, P_E);$
                $\{w_E = \beta \wedge \alpha \underset{P_E}{\Longrightarrow} \beta\}$
```

Figure 3.12: Program for the environment rules P_E

```
$Prog_{\varphi_i}$ :  $R, P_i : Set\ of\ Rule;$
                      $r : Rule;$

                      $\{w_i = \alpha \wedge oldw_i = \alpha\}$
                      $R := \varphi_i(oldw_E, P_i);$
                      $w_i := 0L\_rewrite(w_i, R);$
                      $\{w_i = \beta \wedge \alpha \underset{\varphi_i(w_E)}{\Longrightarrow} \beta \wedge oldw_i = \alpha\}$
```

Figure 3.13: Program for mapping φ_i

```
$Prog_{\psi_i}$ :  $R, R_i : Set\ of\ Rules;$
                   $r : Rule;$

                   $\{w_E = \alpha \wedge w_i = \xi \wedge oldw_i = \delta \wedge \delta \underset{\varphi_i(w_E)}{\Longrightarrow} \xi\}$
                   $R := \psi_i(oldw_i, R_i);$
                   $w_E = parallel\_rewrite(w_E, R);$
                   $\left\{ w_E = \beta \wedge \alpha \underset{\psi_i(oldw_i)}{\Longrightarrow} \beta \wedge w_i = \xi \wedge oldw_i = \delta \wedge \delta \underset{\varphi_i(w_E)}{\Longrightarrow} \xi \right\}$
```

Figure 3.14: Program for mapping ψ_i

3.3 A new formal approach to connect Grammar systems with concurrent programs

3.3.1 Owicki-Gries Theory

This verification strategy is defined for concurrent programs written in DGC language. It assumes the use of First Order Logic as the assertion language for the definition of predicates.

Concurrent execution or multiprogramming means that various sequential programs run simultaneously. Actions change the state of the multiprogram, so the critical question is what happens if two overlapping actions change the same state of the multiprogram in a conflicting manner.

We consider a multiprogram annotated in such a way that the annotation provides a precondition for the multiprogram as a whole and a precondition for each action in each individual program. Then, by Owicki and Gries, this annotation is correct whenever each individual predicate is correct.

To say that the predicate Q in a program is *locally correct* we distinguish two cases:

– If Q is the initial predicate of the program, it is locally correct whenever it is implied by the precondition of the program as a whole. Also we may say that Q satisfies the hypothesis of the problem which is to be solved.

– If Q is preceded by $\{P\}\,\mathcal{S}$, i.e. by atomic action $\mathcal{S}$ with precondition P, it is locally correct whenever $\{P\}\,\mathcal{S}\,\{Q\}$ is a correct Hoare-triple.

A sequential program is *partially locally correct* if all its predicates are locally correct and the last predicate satisfies the requirements of the problem solved, provided that it halts. A sequential program is *totally locally correct* if it is partially correct and always halts.

We say that a predicate Q in a multiprogram $\mathcal{M}$ is *globally correct* whenever for each $\{P\}\,\mathcal{S}$, i.e. for each action $\mathcal{S}$ in program $\mathcal{M}$ with precondition P, $\{P \wedge Q\}\,\mathcal{S}\,\{Q\}$ is a correct Hoare-triple.

To understand how powerful is the concurrent programming paradigm, and also how hard it is to prove global correctness we give this simple example:

Example 1 *Consider this program:*

$$\begin{array}{lrcl} P_1: & x := & y+1; & a \\ & x := & y^2; & b \\ & x := & x-y & c \end{array}$$

If we start in an initial state $\{x = 7 \wedge y = 3\}$, it will deliver $\{x = 6 \wedge y = 3\}$ as a final state.

Example 2 *Let us consider now the following simple program:*

$$\begin{array}{lrcl} P_2: & y := & x+1; & u \\ & y := & x^2; & v \\ & y := & y-x & w \end{array}$$

When started in the same initial state $\{x = 7 \wedge y = 3\}$, it yields $\{x = 7 \wedge y = 42\}$.

Now if we run these programs concurrently we will get 20 possible values for x and y. For instance, one possibility is to run the two programs as follows: a, u, b, v, w, c (the letters represent the program lines) starting from the same state and get the output $\{x = -224, y = 240\}$. While each of the individual programs is of an extreme simplicity, its composition leads to a rather complicated output. For more examples we refer to [BD93].

Here we can see the analogy: the components of a Grammar systems are similar to the simple programs in a multiprogram. In section 3.3.2 we introduce proofs that show how to take advantage of this analogy.

3.3.2 Can Grammar systems benefit from concurrent programming?

We know from [DP97] the following theorem:

Theorem 5 $\{a^n b^n c^n \mid n \geq 0\} \in CD_2(= 2)$.

In [DP97] theorem 5 is proved using the strategy analysis by cases. We transcribe below the proof given in [DP97], in order to emphasize the difference between the use of analysis by cases and our approach.

Proof. Grammar system $\Gamma_1 \in CD_2(= 2)$ is defined in this way:

$$\begin{aligned}\Gamma_1 = (&\{a, b, c\}, (\{S, A, A', B, B'\}, \emptyset, P_1, = 2),\\ &(\{S, A, A', B, B'\}, \{a, b, c\}, P_2, = 2), S)\end{aligned}$$

where:
$P_1 = \{S \rightarrow S, S \rightarrow AB, A' \rightarrow A, B' \rightarrow B\}$,

$P_2 = \{A \rightarrow aA'b, B \rightarrow cB', A \rightarrow ab, B \rightarrow c\}$.

It is proved that $L_{=2}(\Gamma_1) = \{a^n b^n c^n \mid n \geq 0\}$ in the following way:

We have to start from S. Only P_1 can be used. Applying the rule $S \rightarrow S$ twice changes nothing, hence eventually we shall perform the step $S \underset{P_1}{\Longrightarrow} S \underset{P_1}{\Longrightarrow} AB$.

From now on, S will not appear again. Only P_2 can be applied to AB. If we use the nonterminal rules, we get: $AB \underset{P_2}{\Longrightarrow} aA'bB \underset{P_2}{\Longrightarrow} aA'bcB'$.

In general, from a string of the form $a^i A b^j c^k B$ (initially we have $i = j = k = 0$), we can obtain in this way $a^i A b^j c^k B \underset{P_2}{\Longrightarrow} a^{i+1} A' b^{j+1} c^{k+1} B'$.

To such a string we have to apply P_1 again and then we get:

$$a^{i+1} A' b^{j+1} c^{k+1} B' \underset{P_2}{\Longrightarrow} a^{i+1} A b^{j+1} c^{k+1} B.$$

This is the only possibility of using P_1. However, P_2 can be applied to a string $a^i A b^j c^k B$ in the $= 2$ mode using only one nonterminal rule (replacing either A or B by A' or B', respectively), and one terminal rule (removing the remaining symbol A or B). To a string containing only one nonterminal (which is different from S), none of the two components can be applied. Consequently, we have to use, in turn, the first component and the nonterminal rules of the second one, and we have to finish the derivation by using the terminal rules of P_2. ■

```
$MProg_{\Gamma_1}$ :  {$Begin\ Main\ Program$}
        $w : String$;
        $grammar : Integer$;
        $N_1, N_2 : Set\ of\ Character$;

        $w := S$;
        $\mathbf{P} : \{\mathbf{w = S}\}$
        $grammar := Value(1,2)$;
        $do\ Contains(w, N_1 \cup N_2) \rightarrow$
              $Prog_{1,\Gamma_1}\ \|\ Prog_{2,\Gamma_1}$
        $od$;
        $Print(w)$;
        $\mathbf{Q} : \{\mathbf{w} \in \{\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n \mid \mathbf{n} \geq \mathbf{0}\}\}$
        {$End\ Main\ Program$}
```

Figure 3.15: Translation of grammar Γ_1 to multiprogram $MProg_{\Gamma_1}$

```
$Prog_{1,\Gamma_1}$ :  {$program\ for\ G_1$}
        $k, cont : Integer$;
        $P_1 : Set\ of\ Rule$;

        $\{w = \alpha\}$
        $k := 2$;
        $do\ true \rightarrow$
            $if\ grammar = 1 \rightarrow$   $\langle \{w = \alpha \wedge grammar = 1 \wedge k = 2\}$
                                $cont := 0$;
                                $do\ cont \neq k \rightarrow$
                                        $if$   $w = xSy \rightarrow w := xSy$;
                                             $\mid w = xSy \rightarrow w := xABy$;
                                             $\mid w = xA'y \rightarrow w := xAy$;
                                             $\mid w = xB'y \rightarrow w := xBy$;
                                        $fi$;
                                        $cont := cont + 1$;
                                $od$;
                                $grammar := Value(1,2); \rangle$
            $fi$;
        $\{w = \beta \wedge \alpha \overset{=k}{\underset{P_1}{\Longrightarrow}} \beta \wedge grammar = j \wedge k = 2\}$
        $od$;
```

Figure 3.16: Translation of grammar component G_1 from Γ_1 to program $Prog_{1,\Gamma_1}$

```
$Prog_{2,\Gamma_1}$ :  $\{program\ for\ G_2\}$
            $k, cont : Integer;$
            $P_2 : Set\ of\ Rule;$

            $\{w = \alpha\}$
            $k := 2;$
            $do\ true \rightarrow$
                $if\ grammar = 2 \rightarrow$   $\langle\ \{w = \alpha \wedge grammar = 2 \wedge k = 2\}$
                                  $cont := 0;$
                                  $do\ cont \neq k \rightarrow$
                                            $if$    $w = xAy \rightarrow w := xaA'by;$
                                                 $\mid w = xBy \rightarrow w := xcB'y;$
                                                 $\mid w = xAy \rightarrow w := xaby;$
                                                 $\mid w = xBy \rightarrow w := xcy;$
                                            $fi;$
                                            $cont := cont + 1;$
                                  $od;$
                                  $grammar := Value(1, 2); \rangle$
                $fi;$
            $\{w = \beta \wedge \alpha \overset{=k}{\underset{P_2}{\Longrightarrow}} \beta \wedge grammar = j \wedge k = 2\}$
            $od;$
```

Figure 3.17: Translation of grammar component G_2 from Γ_1 to program $Prog_{2,\Gamma_1}$

Below we provide another proof for $\{a^n b^n c^n \mid n \geq 0\} \in CD_2(= 2)$, different from the one presented in theorem 5 and based on the system invariant notion from Owicki-Gries theory. But before we introduce some definitions.

According to the Owicki-Gries Theory the Definition 16 can be equivalently expressed in the following way:

Definition 18 *A Grammar system* $\Gamma \in \{CD, PC, CPC, NPC, NCPC\}$ *with initial configuration* $(S_1, ..., S_n)$, $n \geq 1$, *is equivalent to a DGC multiprogram* $\mathcal{P}$ *iff* $\{(w_1, ..., w_n = S_1, ..., S_n)\}$ $\mathcal{P}$ $\{w_j \in L(\Gamma) \wedge 1 \leq j \leq n\}$ *is globally correct and for all* $w \in L(\Gamma)$ *program* $\mathcal{P}$ *computes* w.

For some problems, like $\{a^n b^n c^n \mid n \geq 0\} \in CD_2(= 2)$ with Grammar system Γ_1 defined as before, it is possible to prove global correctness through an invariant, what is called *system invariant* proof strategy. Applying this strategy requires to find a predicate that remains invariant through all the computation and that synthesizes the behavior of the multiprogram. If such predicate is found the number of proofs reduces to linear.

Definition 19 *(System Invariant) By definition a relation I is a System Invariant whenever:*
- it holds initially, i.e. is implied by the precondition of the multiprogram as a whole,
- it is maintained by each individual atomic statement $\{Q\}\mathcal{S}$ *of each individual component, i.e. whenever for each such* $\{Q\}\mathcal{S}$, $\{I \wedge Q\}$ $\mathcal{S}$ $\{I\}$ *is a correct Hoare-triple.*

Proof. As explained in proposition 1, from Grammar system Γ_1 we construct the equivalent concurrent program $MProg_{\Gamma_1}$ in figure 3.15. Grammars G_1 and G_2 in Γ_1 with $= 2$-mode of derivation correspond in $MProg_{\Gamma_1}$ to program $Prog_{1,\Gamma_1}$ in figure 3.16 and to the program $Prog_{2,\Gamma_1}$ in figure

3.17, respectively. Both programs $Prog_{1,\Gamma_1}$ and $Prog_{2,\Gamma_1}$ run in parallel in $MProg_{\Gamma_1}$ and perform exactly 2 rewritings selected from rewriting rules P_1 and P_2 respectively over the common string w when the variable *grammar* indicates its turn to do so.

We prove global correctness of multiprogram $MProg_{\Gamma_1}$ using the following system invariant:

$$\begin{aligned}\text{Inv } \mathbf{I}: \quad & w = S \vee (w = a^nA\text{'}b^nc^nB\text{'} \wedge n \geq 1) \vee (w = a^nAb^nc^nB \wedge n \geq 0)\vee \\ & \vee(w = a^nA\text{'}b^nc^n \wedge n \geq 1) \vee (w = a^nb^nc^nB\text{'} \wedge n \geq 1)\vee \\ & \vee(w = a^nb^nc^n \wedge n \geq 1)\end{aligned}$$

According to definition 19 proving that the predicate I is invariant is equivalent to proving that:

1. $(w = S) \rightarrow I$,
2. $\{I\}$ *do Contains*$(w, N_1 \cup N_2)$*...od* $\{I\}$ which, according to the loop semantic (Definition 3.2.1), requires to prove the following statements:
   (a) $I \wedge \sim Contains(w, N_1 \cup N_2) \Rightarrow I$,
   (b) $\{I \wedge Contains(w, N_1 \cup N_2)\}\ Prog_{1,\Gamma_1}\ \{I\}$,
   (c) $\{I \wedge Contains(w, N_1 \cup N_2)\}\ Prog_{2,\Gamma_1}\ \{I\}$.

We also want to prove that if the program finishes, it produces one string in $L(\Gamma_1)$:

$$I \wedge (MProg_{\Gamma_1} \text{ terminates}) \longrightarrow (w = a^nb^nc^n \wedge n \geq 1).$$

One possible proof is the following:

1. $(w = S) \rightarrow I$
   $\leftrightarrow$ { Logic property: $p \Rightarrow p \vee q$}
   true
2. $\{I\}$ *do Contains*$(w, N_1 \cup N_2)$*...od* $\{I\}$, which requires to prove that:
   (a) $I \wedge \sim Contains(w, N_1 \cup N_2) \Rightarrow I$
   $\leftrightarrow$ { Logic property: $p \wedge q \Rightarrow p$}
   true

   (b) $\{I \wedge Contains(w, N_1 \cup N_2)\}\ Prog_{1,\Gamma_1}\ \{I\}$
   $\leftrightarrow$ { Replacing, simplifying}
   $\{w = S \vee (w = a^nA\text{'}b^nc^nB\text{'} \wedge n \geq 1) \vee (w = a^nAb^nc^nB \wedge n \geq 0) \vee$
   $\vee (w = a^nA\text{'}b^nc^n \wedge n \geq 1) \vee (w = a^nb^nc^nB\text{'} \wedge n \geq 1)\}$
   $Prog_{1,\Gamma_1}$
   $\{w = S \vee (w = a^nA\text{'}b^nc^nB\text{'} \wedge n \geq 1) \vee (w = a^nAb^nc^nB \wedge n \geq 0) \vee$
   $\vee (w = a^nA\text{'}b^nc^n \wedge n \geq 1) \vee (w = a^nb^nc^nB\text{'} \wedge n \geq 1) \vee (w = a^nb^nc^n \wedge n \geq 1)\}$

– If $(w = a^nA'b^nc^n \wedge n \geq 1) \vee (w = a^nb^nc^nB' \wedge n \geq 1)$ then $Prog_{1,\Gamma_1}$ gets into a busy waiting, therefore the only way to make this statement true is by adding in the guard of $Prog_{1,\Gamma_1}$ the condition $w \neq a^nA'b^nc^n \wedge w \neq a^nb^nc^nB'$. By adding this condition, clearly $Prog_{1,\Gamma_1}$ always terminates.
– If it starts in a state satisfying $w = S \vee (w = a^nA'b^nc^nB' \wedge n \geq 1)$ it finishes in a state satisfying $w = S \vee (w = a^nAb^nc^nB \wedge n \geq 0)$.
– And if it starts in a state satisfying $(w = a^nAb^nc^nB \wedge n \geq 0)\vee$
$(w = a^nA'b^nc^n \wedge n \geq 1) \vee (w = a^nb^nc^nB' \wedge n \geq 1)$ if finishes in the same state.

When constructing formal proofs of programs it is not rare to add extra code in order to prove correctness. Examples can be found in [OG76]. We introduce in figure 3.19 a new program $Prog_{1,\Gamma_1}$ that differs from program $Prog_{1,\Gamma_1}$ in figure 3.16 in the new guard $cont \neq 2 \wedge w \neq a^nA'b^nc^n \wedge w \neq a^nb^nc^nB$ for the loop. Because variable n counts the number of occurrences of a's which is equal to the number of b's and c's, n has to be incremented each time new a's are added. $Prog_{2,\Gamma_1}$ in figure 3.17 is modified with this purpose, getting a new program $Prog_{2,\Gamma_1}$ which is shown in figure 3.20. Main Program $MProg_{\Gamma_1}$ is modified adding the declaration of the new variable n which is initialized to zero, producing the program shown in figure 3.18.

(c) $\{I \wedge Contains(w, N_1 \cup N_2)\}$ $Prog_{2,\Gamma_1}$ $\{I\}$.
$Prog_{2,\Gamma_1}$ always terminates.
– If it starts in a state satisfying $w = a^nAb^nc^nB \wedge n \geq 0$ it finishes in a state satisfying $w = a^nA'b^nc^nB' \wedge n \geq 1$.
– And if it starts in a state satisfying $w = S \vee (w = a^nA'b^nc^nB' \wedge n \geq 1)\vee$
$(w = a^nb^nc^nB' \wedge n \geq 1) \vee (w = a^nA'b^nc^n \wedge n \geq 1)$ if finishes in the same state.

We also want to prove that if the program finishes, it computes one string in $L(\Gamma_1)$:

$$I \wedge (MProg_{\Gamma_1} \text{ terminates}) \longrightarrow (w = a^nb^nc^n \wedge n \geq 1).$$

$I \wedge (MProg_{\Gamma_1}$ terminates$) \longrightarrow (w = a^nb^nc^n \wedge n \geq 1)$
$\leftrightarrow$ { Replacing }
$I\wedge \sim Contains(w, N_1 \cup N_2) \longrightarrow w = a^nb^nc^n \wedge n \geq 1$
$\leftrightarrow$ { Replacing, simplifying }
$\{w = a^nb^nc^n \wedge n \geq 1 \longrightarrow w = a^nb^nc^n \wedge n \geq 1\}$
$\leftrightarrow \{ p \longrightarrow p\}$
true

This proves that $\{w = S\}$ $MProg_{\Gamma_1}$ $\{w = a^nb^nc^n \wedge n \geq 1\}$ is globally correct. The proof that for all $w \in \{a^nb^nc^n \mid n \geq 0\}$ there is a string w computed by program $MProg_{\Gamma_1}$ is based on the alternative construction *if* of program $Prog_{2,\Gamma_1}$. This alternative command chooses nondeterministically between replacing symbol A in w for string $aA'b$ or for string ab; or replacing symbol B for strings bB' or b. Therefore all the strings $w = a^nb^nc^n$ with $n \geq 1$ number of a's, b's and c's can be generated. This completes the proof of $\{a^nb^nc^n \mid n \geq 0\} \in CD_2(= 2)$. ■

```
$MProg_{\Gamma_1}$ :  $\{Begin\ Main\ Program\}$
              $w : String;$
              $n : int;$
              $grammar : Integer;$
              $N_1, N_2 : Set\ of\ Character;$

              $w, n := S, 0;$
              $\mathbf{P} : \{\mathbf{w = S}\}$
              $grammar := Value(1, 2);$
              $do\ Contains(w, N_1 \cup N_2) \rightarrow$
                  $Prog_{1,\Gamma_1} \parallel Prog_{2,\Gamma_1}$
              $od;$
              $Print(w);$
              $\mathbf{Q} : \{\mathbf{w} \in \{\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n \mid \mathbf{n} \geq \mathbf{0}\}\}$
              $\{End\ Main\ Program\}$
```

Figure 3.18: Annotated program $MProg_{\Gamma_1}$ resulting from global correctness proof

```
$Prog_{1,\Gamma_1}$ :  $\{program\ for\ G_1\}$
            $k, cont : Integer;$
            $P_1 : Set\ of\ Rule;$

            $\{w = \alpha\}$
            $k := 2;$
            $do\ true \rightarrow$
                $if\ grammar = 1 \rightarrow$  $\langle\ \{w = \alpha \wedge grammar = 1 \wedge k = 2\}$
                                  $cont := 0;$
                                  $do\ cont \neq k \wedge w \neq a^nA'b^nc^n \wedge w \neq a^nb^nc^nB' \rightarrow$
                                          $if$   $w = xSy \rightarrow w := xSy;$
                                                 $\mid w = xSy \rightarrow w := xABy;$
                                                 $\mid w = xA'y \rightarrow w := xAy;$
                                                 $\mid w = xB'y \rightarrow w := xBy;$
                                          $fi;$
                                          $cont := cont + 1;$
                                  $od;$
                                  $grammar := Value(1, 2);\ \rangle$
                $fi;$
            $\{w = \beta \wedge \alpha \overset{=k}{\underset{P_1}{\Longrightarrow}} \beta \wedge grammar = j \wedge k = 2\}$
            $od;$
```

Figure 3.19: Annotated program $Prog_{1,\Gamma_1}$ resulting from global correctness proof

```
$Prog_{2,\Gamma_1}$ :  $\{program\ for\ G_2\}$
          $k, cont : Integer;$
          $P_2 : Set\ of\ Rule;$

          $\{w = \alpha\}$
          $k := 2;$
          $do\ true \rightarrow$
              $if\ grammar = 2 \rightarrow$  $\langle$ $\{w = \alpha \wedge grammar = 2 \wedge k = 2\}$
                                      $cont := 0;$
                                      $do\ cont \neq k \rightarrow$
                                           $if$   $w = xAy \rightarrow w, n := xaA'by, n + 1;$
                                                 $|\ w = xBy \rightarrow w := xcB'y;$
                                                 $|\ w = xAy \rightarrow w, n := xaby, n + 1;$
                                                 $|\ w = xBy \rightarrow w := xcy;$
                                           $fi;$
                                           $cont := cont + 1;$
                                      $od;$
                                      $grammar := Value(1, 2);$ $\rangle$
              $fi;$
          $\{w = \beta \wedge \alpha \overset{=k}{\underset{P_2}{\Longrightarrow}} \beta \wedge grammar = j \wedge k = 2\}$
          $od;$
```

Figure 3.20: Annotated program $Prog_{2,\Gamma_1}$ resulting from global correctness proof

Owicki-Gries logic of programming, as analysis by cases, contemplates the way to prove some matters related with dynamic aspects of a grammar such as: reachability of a configuration, absence of progress because of circularity (in the case of PC Grammar systems with communication by query), situations of deadlock, termination, etc.

When considering progress the only instruction in the DGC language that can stop the progress of a program is the alternative construction. In case all the guards of an *if* are *false* it waits until some guard becomes *true*.
So we can state the rule of progress in the following way:

Definition 20 *(Rule of Progress) Statement* $\{Q\}$ ***if*** $B_1 \rightarrow S_1, ..., B_{1n} \rightarrow S_n$ ***fi*** *in a component is guaranteed to terminate iff the rest of the system, when constrained to Q, will converge to a state in which* $B_1 \vee ... \vee B_n$ *is stably true in a finite number of steps .*

From the annotated program $MProg_{\Gamma_1}$ in figure 3.18 which proved to behave like the Grammar system Γ_1 the *progress* of programs $Prog_{1,,\Gamma_1}$ and $Prog_{2,,\Gamma_1}$, or equivalently grammars G_1 and G_2, can be deduced:

Corollary 1 *Programs* $Prog_{1,,\Gamma_1}$ *and* $Prog_{2,,\Gamma_1}$ *satisfy the rule of progress.*

Proof. With respect to program $Prog_{1,\Gamma_1}$ we need to prove for the alternative construction that the following statement terminates:

```
$\{I \wedge cont \neq 2 \wedge w \neq a^nA'b^nc^n \wedge w \neq a^nb^nc^nB'\}$
    $if$     $w = xSy \rightarrow w := xSy;$
           $| w = xSy \rightarrow w := xABy;$
           $| w = xA'y \rightarrow w := xAy;$
           $| w = xB'y \rightarrow w := xBy;$
    $fi;$
    $cont := cont + 1;$
```

This is equivalent to proving that program $MProg_{\Gamma_1}$ when constrained to $[I \wedge cont \neq 2 \wedge w \neq a^nA'b^nc^n \wedge w \neq a^nb^nc^nB']$ will, in a finite number of steps, converge to a state in which $[w = xSy \vee w = xA'y \vee w = xB'y]$ is stably *true*. This is satisfied, because the following statement evaluates to *true*: $[I \wedge cont \neq 2 \wedge w \neq a^nA'b^nc^n \wedge w \neq a^nb^nc^nB'] \Rightarrow [w = xSy \vee w = xA'y \vee w = xB'y]$.

Following the same reasoning progress of program $Prog_{2,\Gamma_1}$ is guaranteed because when it is constrained to $[I \wedge cont \neq 2]$ in a finite number of steps it converges to state $[w = xAy \vee w = xBy]$. ■

Termination is for some programs a desirable property. We can prove that the termination of program $MProg_{\Gamma_1}$ can not be guaranteed, or equivalently that the derivation of a string of terminal symbols in Γ_1 can not be guaranteed.

Corollary 2 *Program $MProg_{\Gamma_1}$ does not necessarily halt.*

Proof. Two possible situations can occur:

1. For each loop execution the alternative construction *if* from program $Prog_2$ indefinitely selects the guarded instructions that replace symbol A for string $aA'b$ and symbol B for string bB'. In this way no terminal string $w = a^nb^nc^n$ is ever obtained.

2. The following instance of executions can take place:
   $\{w = a^{n-1}A'b^{n-1}c^{n-1}B' \wedge n \geq 01\}$
   $Prog_{1,\Gamma_1}$
   $\{w = a^{n-1}Ab^{n-1}c^{n-1}B \wedge n \geq 0\}$
   $Prog_{2,\Gamma_1}$
   $\{(w = a^nA'b^nc^n \wedge n \geq 1) \vee (w = a^nb^nc^nB' \wedge n \geq 1)\}$
   $Prog_{1,\Gamma_1} \quad || \quad Prog_{2,\Gamma_1}$
   $\{(w = a^nA'b^nc^n \wedge n \geq 1) \vee (w = a^nb^nc^nB' \wedge n \geq 1)\}$
   From this state of the system program $Prog_{1,\Gamma_1}$ and program $Prog_{2,\Gamma_1}$ do not perform any action, but they keep evaluating the guards of the loop, getting the value *false*. This is equivalent to performing a skip instruction, so that the state of the system can no longer change: $\{(w = a^nA'b^nc^n \wedge n \geq 0) \vee (w = a^nb^nc^nB' \wedge n \geq 0)\}$.

■

According to proposition 1 we can automatically translate a CD Grammar system Γ to a concurrent program $\mathcal{P}_\Gamma$. In this way the problem of proving that Γ generates a language L is transformed into the problem of first proving that the program $\mathcal{P}_\Gamma$ obtained from the translation is correct with respect to the precondition $\{w = S\}$ and the postcondition $\{w \in L\}$ and second that $\mathcal{P}_\Gamma$ generates all the strings in L. Therefore besides analysis by cases the *global correctness* strategy from Owicki-Gries Theory can be also used to prove that a Grammar system generates a language. It is not only the generation of a given language but also derivation properties like starvation, deadlock, progress, termination, etc, formulated for Grammar systems that can be equivalently expressed in the programming framework. Corollaries 1 and 2 exemplify the use of this strategy for proving the properties of progress and termination of a Grammar system.

From the previous analysis we can remark on some advantages of the *system invariant strategy* over the *analysis by cases technique*:

- With analysis by cases we can capture the system global behavior by a general sequence of derivations *including detailed information,* such as how grammars interact, which productions they apply, how they change the sentential form, etc. When we apply the system invariant technique we capture the global behavior of the system by an invariant that shows all possible values of the sentential form and hides information that analysis by cases gives. So we can say that the invariant system captures global behavior in a *more abstract way*.

- With analysis by cases, apart from showing the shape that any sequence of derivation should have, we need to prove that this is the only possible sequence of derivations, adding an explanation in *natural language*. Equivalently with the system invariant technique we need to prove in the framework of logic that each program $Prog_i$ preserves the invariant, which is achieved by *formal proofs*.

Another advantage of the Owicki-Gries logic of programming is that it can help to simultaneously construct the Grammar system that generates a given language specification and the proof that the grammar generates the language. This is a great improvement, because there is no technique in the framework of Grammar systems, to help us to solve this kind of problem. We present theorems 6, 7 and 8 to exemplify the use of this strategy.

Theorem 6 $L_{cd} \in NPC_3(Reg)$

Proof. We want to find a non-returning, non-centralized Parallel Communicating Grammar system Γ_2 with regular components that generates L_{cd}. This problem is transformed into the problem of finding a multiprogram $MProg_{\Gamma_2}$ that is equivalent to Γ_2 and globally correct with respect to the specification:

$$\{(w_1 = S_1) \wedge (w_2 = S_2) \wedge (w_3 = S_3)\}\ MProg_{\Gamma_2}\ \{w_1 \in L_{cd}\}$$

Besides it has to be proved that for all $w \in L_{cd}$ program $MProg_{\Gamma_2}$ computes w.

The problem remains the same, but we change the tools to solve it: instead of *analysis by cases* available in the framework of Grammar systems we use *Logic, Owicki-Gries Theory* and *programming strategies* from the Programming framework.

The strategy we follow for this proof is called *refinement of the problem* and it is frequently used for the development of programs. It consists on:

(I) First, start with an outline of the solution, which identifies the basic principle by which the input can be transformed into the output. Define pre and post conditions for each of the subproblems that are identified as part of the solution for the whole problem.

For our problem we propose this idea:
$\{(w_1 = S_1) \wedge (w_2 = S_2) \wedge (w_3 = S_3)\}$
Subproblem 1*: (Rewrite)^p, with $p \geq 1$*
$\left\{(w_1 = S_1) \wedge (w_i = a^p S_i) \wedge (w_j = c^p S_j) \wedge (p \geq 1) \wedge \{i, j\} = \{2, 3\}\right\}$
Subproblem 2*: (Rewrite; Communication)$^+$*
Find a way to stop the production of *a's* and *c's*, through synchronization by communication.
$\{(w_1 = a^r N_1) \wedge (w_j = c^r N_2) \wedge (r \geq 1) \wedge (N_1, N_2 \in N) \wedge j \in \{2, 3\}\}$
Subproblem 3*: (Rewrite)^m, with $m \geq 1$*
$\left\{ (w_1 = a^r b^m Q_k) \wedge (w_j = c^r d^{m-1} N_3) \wedge (r, m \geq 1) \wedge (Q_k \in K) \wedge (N_3 \in N) \wedge j \in \{2, 3\} \right\}$
Subproblem 4*: Communication*
$\left\{(w_1 = a^r b^m c^r d^{m-1} N_3) \wedge (r, m \geq 1) \wedge (N_3 \in N)\right\}$
Subproblem 5*: Rewrite*
$\{(w_1 = a^r b^m c^r d^m) \wedge (r, m \geq 1)\}$
or equivalently
$\{w_1 \in \{a^r b^m c^r d^m \wedge r \geq 1 \mid m \geq 1\}\}$

(II) Make precise the outline from (I), refine the subproblems trying to find simultaneously the instructions that solve the subproblems and the proof of its local correctness.

In the refinement of subproblems 1 to 5 we propose three programs $Prog_{1,\Gamma_2}$, $Prog_{2,\Gamma_2}$ and $Prog_{3,\Gamma_2}$. These programs form the multiprogram $MProgram_{\Gamma_2}$, run simultaneously behaving like a non-returning, non-centralized Grammar system with regular productions and behave locally correctly with respect to the subproblems that we have identified in the previous step.
In the case of Subproblem 1 we propose this refinement:

$\{(w_1 = S_1) \wedge (w_2 = S_2) \wedge (w_3 = S_3)\}$
Subproblem 1*: Rewrite^n, with $n \geq 1$*
$Prog_{1,\Gamma_2}$ rewrites $n-1$ times S_1 to aS_1 and then rewrites S_1 to aA, $Prog_{2,\Gamma_2}$ rewrites $n-1$ times S_2 to cS_2 and then rewrites S_2 to cB. $Prog_{3,\Gamma_2}$ rewrites $n-1$ times S_3 to S_3, until it decides to finish the production of a's and c's, rewriting S_3 to Q_2.

To be sure that $w_2 = c^n B$ when $Prog_{3,\Gamma_2}$ introduces Q_2, $Prog_{3,\Gamma_2}$ should not be able to rewrite S_2, and after $Prog_{2,\Gamma_2}$ introduces B it should rewrite it for other nonterminal and not introduce B again.

The reason why $w_1 = a^n A$ and $w_1 \neq a^n S_1$ is that this is the only possibility that does not lead to deadlock, as the states of the next subproblem show.
$\{(w_1 = a^n A) \wedge (w_2 = c^n B) \wedge (w_3 = Q_2) \wedge (n \geq 1)\}$

For Subproblem 2 we propose this sequence of rewritings and communications as a refinement:

$\{(w_1 = a^n A) \wedge (w_2 = c^n B) \wedge (w_3 = Q_2) \wedge (n \geq 1)\}$

Subproblem 2
- *Communication*
- $\{w_1 = a^n A \wedge w_2 = c^n B \wedge w_3 = c^n B \wedge n \geq 1\}$
- *Rewrite*
- $Prog_{1,\Gamma_2}$ rewrites A to A', $Prog_{2,\Gamma_2}$ rewrites B to Q_1 and $Prog_{3,\Gamma_2}$ rewrites B to D
- We do not allow any possibility other than $w_1 = a^n A\text{'} \wedge w_2 = c^n Q_1 \wedge w_3 = c^n D$.
- To be sure that $w_1 = a^n A$' after the rewriting step, we need $Prog_{2,\Gamma_2}$ to be defined only for A', and after $Prog_{1,\Gamma_2}$introduces A' it should rewrite it to other nonterminal and do not introduce A' again.
- $\{w_1 = a^n A\text{'} \wedge w_2 = c^n Q_1 \wedge w_3 = c^n D \wedge n \geq 1\}$
- *Communication*

$\{(w_1 = a^n A\text{'}) \wedge (w_2 = c^n a^n A\text{'}) \wedge (w_3 = c^n D) \wedge (n \geq 1)\}$

In the case of Subproblem 3 this is a possible refinement:

$\{(w_1 = a^n A\text{'}) \wedge (w_2 = c^n a^n A\text{'}) \wedge (w_3 = c^n D) \wedge (n \geq 1)\}$

Subproblem 3*: Rewrite*$^{m+1}$*, with* $m \geq 1$

$Prog_{1,\Gamma_2}$ rewrites A' to A" and rewrites $m-1$ times A" to bA", and then rewrites A" to bQ_3. $Prog_{2,\Gamma_2}$ always rewrites A' to A' and $Prog3$ rewrites D to D', then D' to D" and rewrites $m-1$ times D" to dD"

$\left\{(w_1 = a^n b^m Q_3) \wedge (w_2 = c^n a^n A\text{'}) \wedge (w_3 = c^n d^{m-1} D\text{''}) \wedge (n, m \geq 1)\right\}$

The refinements for Subproblem 4 and Subproblem 5 are very simple:

$\left\{(w_1 = a^n b^m Q_3) \wedge (w_2 = c^n a^n A\text{'}) \wedge (w_3 = c^n d^{m-1} D\text{''}) \wedge (n, m \geq 1)\right\}$

Subproblem 4: *Communication*

$\left\{(w_1 = a^n b^m c^n d^{m-1} D\text{''}) \wedge (w_2 = c^n a^n A\text{'}) \wedge (w_3 = c^n d^{m-1} D\text{''}) \wedge (n, m \geq 1)\right\}$

Subproblem 5: *Rewrite*

$Prog_{1,\Gamma_2}$ rewrites D" to d

$\{(w_1 \in \{a^n b^m c^n d^m) \wedge (n, m \geq 1)\}\}$

From the previous analysis we propose the multiprogram $MProg_{\Gamma_2}$ (Figure 3.21) with concurrent programs $Prog_{1,\Gamma_2}$ (Figure 3.22), $Prog_{2,\Gamma_2}$ (Figure 3.23) and $Prog_{3,\Gamma_2}$ (Figure 3.24) running concurrently.

(III) Prove global correctness of multiprogram $MProg_{\Gamma_2}$.

In this case we have to show using Owicki-Gries Theory that the multiprogram $MProg_{\Gamma_2}$ we constructed satisfies the next specification:

$$\{(w_1 = S_1) \wedge (w_2 = S_2) \wedge (w_3 = S_3)\} MProg_{\Gamma_2} \{w_1 \in L_{cd}\}$$

We find the following invariant that guarantees the proof of global correctness:

$$\mathit{Inv}\ \mathbf{I}: \left[\begin{array}{c} (w_1 = a^n S_1 \wedge n \geq 0) \vee (w_1 = a^n A \wedge n \geq 1) \vee (w_1 = a^n A\text{'} \wedge n \geq 1) \vee \\ \vee (w_1 = a^v b^n A\text{''} \wedge v \geq 1 \wedge n \geq 0) \vee (w_1 = a^v b^n Q_3 \wedge v \geq 1 \wedge n \geq 1) \vee \\ \vee (w_1 = a^v b^n c^g d^h D\text{''} \wedge v, n, g \geq 1 \wedge h \geq 0) \vee \\ \vee (w_1 = a^e b^f c^g d^h \wedge e, f, g, h \geq 1) \end{array}\right] \wedge$$
$$\wedge \left[\begin{array}{c} (w_2 = c^q S_2 \wedge q \geq 0) \vee (w_2 = c^q B \wedge q \geq 1) \\ (w_2 = c^q Q_1 \wedge q \geq 1) \vee (w_2 = c^q a^r A\text{'} \wedge q, r \geq 1) \end{array}\right] \wedge$$
$$\wedge \left[\begin{array}{c} (w_3 = S_3) \vee (w_3 = Q_2) \vee (w_3 = c^n B \wedge n \geq 1) \vee (w_3 = c^n D \wedge n \geq 1) \vee \\ \vee (w_3 = c^n D\text{'} \wedge n \geq 1) \vee (w_3 = c^n d^m D\text{''} \wedge n \geq 1 \wedge m \geq 0) \end{array}\right]$$

For all $w \in L_{cd}$ program $MProg_{\Gamma_2}$ computes w because program $Prog_{1,\Gamma_2}$ arbitrarily chooses when to introduce query symbols that produce communication, stopping the production of the same number of a's and c's, and the same number of b's and d's.

IV) As explained in proposition 2 the non-returning, non-centralized Parallel Communicating Grammar system Γ_2 with three regular components is equivalent to multiprogram $MProg_{\Gamma_2}$ (Figure 3.21) with concurrent programs $Prog_{1,\Gamma_2}$ (Figure 3.22), $Prog_{2,\Gamma_2}$ (Figure 3.23) and $Prog_{3,\Gamma_2}$ (Figure 3.24) running concurrently. Γ_2 is defined in this way:

$$\Gamma_2 = (N, K, \{a, b, c, d\}, (P_1, S_1), (P_2, S_2), (P_3, S_3))$$

where:

$N = \{S_1, S_2, S_3, A, A\text{'}, A\text{''}, B, D, D\text{'}, D\text{''}\}$

$K = \{Q_1, Q_2, Q_3\}$

$P_1 = \{S_1 \longrightarrow aS_1, S_1 \longrightarrow aA, A \longrightarrow A\text{'}, A\text{'} \longrightarrow A\text{''}, A\text{''} \longrightarrow bA\text{''}, A\text{''} \longrightarrow bQ_3, D\text{''} \longrightarrow d\}$

$P_2 = \{S_2 \longrightarrow cS_2, S_2 \longrightarrow cB, B \longrightarrow Q_1, A\text{'} \longrightarrow A\text{'}\}$

$P_3 = \{S_3 \longrightarrow S_3, S_3 \longrightarrow Q_2, B \longrightarrow D, D \longrightarrow D\text{'}, D\text{'} \longrightarrow D\text{''}, D\text{''} \longrightarrow dD\text{''}\}$ ■

```
$MProg_{\Gamma_2}$ :  $N, K, T : Set\ of\ String$;
                 $w_1, w_2, w_3 : String$;

                 $N := \{S_1, S_2, S_3, A, A', A'', B, D, D', D''\}$;
                 $K := \{Q_1, Q_2, Q_3\}$;
                 $w_1, w_2, w_3 := S_1, S_2, S_3$;
                 $do\ Contains(w_1, N) \vee Contains(w_2, N) \vee Contains(w_3, N) \rightarrow$
                        $Prog_{1,\Gamma_2} \parallel Prog_{2,\Gamma_2} \parallel Prog_{3,\Gamma_2}$;
                        $if\quad contains(w_1, K) \vee contains(w_2, K) \vee contains(w_3, K) \rightarrow$
                                                              $Communicate(w_1, w_2, w_3)$;
                          $\mid\ \neg(contains(w_1, K) \vee contains(w_2, K) \vee contains(w_3, K)) \rightarrow\ skip$;
                        $fi$;
                 $od$;
                 $Print(w_1)$
```

Figure 3.21: Multiprogram for Γ_2

```
$Prog_{1,\Gamma_2}$ :
          $if$   $w_1 = \alpha S_1\beta$          $\rightarrow w_1 := \alpha a S_1\beta$;
                 $\mid w_1 = \alpha S_1\beta$       $\rightarrow w_1 := \alpha a A\beta$;
                 $\mid w_1 = \alpha A\beta$         $\rightarrow w_1 := \alpha A'\beta$;
                 $\mid w_1 = \alpha A'\beta$        $\rightarrow w_1 := \alpha A''\beta$;
                 $\mid w_1 = \alpha A''\beta$       $\rightarrow w_1 := \alpha b A''\beta$;
                 $\mid w_1 = \alpha A''\beta$       $\rightarrow w_1 := \alpha b Q_3\beta$;
                 $\mid w_1 = \alpha D''\beta$       $\rightarrow w_1 := \alpha d\beta$;
                 $\mid \left( \begin{array}{l} w_1 \neq \alpha S_1\beta \wedge w_1 \neq \alpha A\beta \wedge \\ w_1 \neq \alpha A'\beta \wedge w_1 \neq \alpha A''\beta \wedge \\ w_1 \neq \alpha D''\beta \end{array} \right)$   $\rightarrow skip$;
          $fi$;
```

Figure 3.22: Program for grammar component G_1 in Γ_2

```
$Prog_{2,\Gamma_2}$ :
          $if$   $w_2 = \alpha S_2\beta$          $\rightarrow w_2 := \alpha c S_2\beta$;
                 $\mid w_2 = \alpha S_2\beta$       $\rightarrow w_2 := \alpha c B\beta$;
                 $\mid w_2 = \alpha B\beta$         $\rightarrow w_2 := \alpha Q_1\beta$;
                 $\mid w_2 = \alpha A'\beta$        $\rightarrow w_2 := \alpha A'\beta$;
                 $\mid \neg \left( \begin{array}{l} w_2 = \alpha S_2\beta \vee w_2 = \alpha B\beta \vee \\ w_1 = \alpha A'\beta \end{array} \right)$   $\rightarrow skip$;
          $fi$;
```

Figure 3.23: Program for grammar component G_2 in Γ_2

```
$Prog_{3,\Gamma_2}$ :
        $if$   $w_3 = \alpha S_3\beta$              $\rightarrow w_3 := \alpha S_3\beta;$
             $| w_3 = \alpha S_3\beta$            $\rightarrow w_2 := \alpha Q_2\beta;$
             $| w_3 = \alpha B\beta$              $\rightarrow w_3 := \alpha D\beta;$
             $| w_3 = \alpha D\beta$              $\rightarrow w_3 := \alpha D'\beta;$
             $| w_3 = \alpha D'\beta$             $\rightarrow w_3 := \alpha D''\beta;$
             $| w_3 = \alpha D''\beta$            $\rightarrow w_3 := \alpha dD''\beta;$
             $| \neg \left( \begin{array}{l} w_3 = \alpha S_3\beta \vee w_3 = \alpha B\beta \vee \\ w_3 = \alpha D\beta \vee w_3 = \alpha D'\beta \vee \\ w_3 = \alpha D''\beta \end{array} \right)$   $\rightarrow skip;$
        $fi;$
```

Figure 3.24: Program for grammar component G_3 in Γ_2

The strategy we have presented to solve Theorem 6 differs from the traditional approach not in complexity, since the number of cases considered in the proofs is the same, but in the way we reason about the problem. We suggest that the Owicki-Gries methodology could provide more possibilities for reasoning about problems, in comparison with the strategies commonly used so far in the Grammar system framework because:

1. When a Grammar system Γ can be translated to DGC language, then Owichi-Gries Theory can be used to reason about a derivation problem in terms of logic.

2. It allows reasoning in a forward or data-driven way, similar to the analysis by cases technique, but also in a backward or goal-directed way. The notion of backward reasoning comes from psychology, as is pointed out in [KC58] where this description of problem solving occurs:

   "We may have a choice between starting with where we wish to end, or starting with where we are at the moment. In the first instance we start by analyzing the goal. We ask, "Suppose we did achieve the goal, how would things be different-what subproblems would we have solved, etc.?". This in turn would determine the sequence of problems, and we would work back to the beginning. In the second instance we start by analyzing the present situation, see the implications of the given conditions and lay-out, and attack the various subproblems in a *forward direction*."

3. The division of problems into subproblems is possible because of the semantics of sequential composition:

$$\{P\}\ S_0; S_1\ \{R\} \leftrightarrow \exists\ Q \text{ such that } \{P\}\ S_0\ \{Q\} \wedge \{Q\}\ S_1\ \{R\}$$

   Also goals and subgoals are discussed in the psychology text mentioned above [KC58]:

   " The person perceives in his surrounding goals capable of removing his needs and fulfilling his desires [...] And there is the important phenomenon of emergence of subgoals. The pathways to goals are often perceived as organized into a number of subparts, each of which constitutes an intermediate subgoal to be attained on the way to the ultimate goal."

These characteristics suggest that Owicki-Gries strategies are closer to the human way of reasoning.

We introduce theorem 7 that shows the combined use of the Owicki-Gries strategy, the technique called refinement of problems that we explained in Theorem 6, and the rule of orthogonality that we state below:

Definition 21 *(Rule of Orthogonality) An assertion is maintained by all assignments to variables not occurring in it.*

Theorem 7 $\{xcxc \mid x \in \{a,b\}^*\} \in CCPC(CF)$.

Proof. We are looking for a CCPC Grammar system that generates $\{xcxc \mid x \in \{a,b\}^*\}$

(I) A possible division into subproblems could be:
$\{w_1 = S_1 \wedge w_2 = S_2 \wedge w_3 = S_3 \wedge w_4 = S_4\}$
Subproblem 1: (Rewrite)$^{n+1}$, $n \geq 0$
$Prog_4$ generates $xX, x \in \{a,b\}^*, \mid x \mid= n$.
$\{w_1 = S_1\text{'}\wedge w_2 = S_2\text{'}\wedge w_3 = S_3\text{'}\wedge w_4 = xX \wedge x \in \{a,b\}^* \wedge \mid x \mid= n \wedge X \in N\}$
Subproblem 2: *Communication*
$Prog_4$ communicates with $Prog_2$ and $Prog_3$ sending two copies of w_4.
$\{w_1 = S_1\text{'}\wedge w_2 = xX \wedge w_3 = xX \wedge w_4 = S_4 \wedge x \in \{a,b\}^* \wedge \mid x \mid= n \wedge X \in N\}$
Subproblem 3: $(Rewrite)^p, p \geq 0$
$Prog_2$ and $Prog_3$ replace X by c.
$\{w_1 = S_1\text{'}\wedge w_2 = xc \wedge w_3 = xc \wedge w_4 = yX \wedge x \in \{a,b\}^* \wedge \mid x \mid= n \wedge$
$\wedge X \in N \wedge y \in \{a,b\}^* \wedge \mid y \mid= p\}$
Subproblem 4: *Communication*
$Prog_1$ receives the content of w_2 and w_3 from $Prog_2$ and $Prog_3$.
$\{w_1 = xcxc \wedge w_2 = yX \wedge w_3 = yX \wedge w_4 = S_4 \wedge x \in \{a,b\}^* \wedge \mid x \mid= n \wedge$
$\wedge X \in N \wedge y \in \{a,b\}^* \wedge \mid y \mid= p\}$

From this analysis we simultaneously propose a multiprogram $MProg_{\Gamma_3}$ (Figure 3.25) with programs $Prog_{1,\Gamma_3}$ (Figure 3.26), $Prog_{2,\Gamma_3}$ (Figure 3.27), $Prog_{3,\Gamma_3}$ (Figure 3.28) and $Prog_{4,\Gamma_3}$ (Figure 3.29).

(II) We need to prove the global correctness of the specification
$\{w_1 = S_1 \wedge w_2 = S_2 \wedge w_3 = S_3 \wedge w_4 = S_4\}MProgr_{\Gamma_3}\{w_1 \in \{xcxc \mid x \in \{a,b\}^*\}\}$.
According to the analysis presented in (I) we need to prove the following:

```
$\{[(w_1, w_2, w_3, w_4) = (S_1, S_2, S_3, S_4) \vee (w_1, w_2, w_3, w_4) = (S'_1, xX, xX, S_4)] \wedge x \in \{a, b\}^* \wedge X \in N\}$
$Prog_{1,\Gamma_3} \parallel Prog_{2,\Gamma_3} \parallel Prog_{3,\Gamma_3} \parallel Prog_{4,\Gamma_3}$
$\{[(w_1, w_2, w_3, w_4) = (S'_1, S'_2, S'_3, xX) \vee (w_1, w_2, w_3, w_4) = (S'_1, xc, xc, yX)] \wedge x, y \in \{a, b\}^* \wedge X \in N\}$
$i := 1;$
$do\ i \leq 3 \rightarrow$
        $if \quad \Delta(i) \neq \lambda \rightarrow w_i := \Delta(i);$
            $| \Delta(i) = \lambda \wedge \delta(i) = \lambda \rightarrow skip;$
            $| \Delta(i) = \lambda \wedge \delta(i) \neq \lambda \rightarrow w_i := S_i;$
  $fi;$
  $i := i + 1;$
$od;$
$\{[(w_1, w_2, w_3, w_4) = (S'_1, xX, xX, S_4) \vee (w_1, w_2, w_3, w_4) = (xcxc, yX, yX, S_4)] \wedge x, y \in \{a, b\}^* \wedge X \in N\}$
```

Based on the rule of orthogonality the proof of global correctness of:

$\{[(w_1, w_2, w_3, w_4) = (S_1, S_2, S_3, S_4) \vee (w_1, w_2, w_3, w_4) = (S'_1, xX, xX, S_4)] \wedge x \in \{a, b\}^* \wedge X \in N\}$
$Prog_{1,\Gamma_3} \parallel Prog_{2,\Gamma_3} \parallel Prog_{3,\Gamma_3} \parallel Prog_{4,\Gamma_3}$
$\{[(w_1, w_2, w_3, w_4) = (S'_1, S'_2, S'_3, xX) \vee (w_1, w_2, w_3, w_4) = (S'_1, xc, xc, yX)] \wedge x, y \in \{a, b\}^* \wedge X \in N\}$

reduces to the following proofs of global correctness:

$\{[w_1 = S_1 \vee w_1 = S'_1]\}$
$Prog_{1,\Gamma_3}$
$\{w_1 = S'_1\}$
$\{[w_2 = S_2 \vee w_2 = xX] \wedge x \in \{a, b\}^* \wedge X \in N\}$
$Prog_{2,\Gamma_3}$
$\{[w_2 = S'_2 \vee w_2 = xc] \wedge x \in \{a, b\}^* \wedge X \in N\}$
$\{[w_3 = S_3 \vee w_3 = xX] \wedge x \in \{a, b\}^* \wedge X \in N\}$
$Prog_{3,\Gamma_3}$
$\{[w_3 = S'_3 \vee w_3 = xc] \wedge x \in \{a, b\}^* \wedge X \in N\}$
$\{w_4 = S_4\}$
$Prog_{4,\Gamma_3}$
$\{w_4 = xX \wedge x \in \{a, b\}^* \wedge X \in N\}$

The local correctness of the above specifications is guaranteed by the analysis presented in (I) and by the definition of programs $Prog_{1,\Gamma_3}$ (Figure 3.26), $Prog_{2,\Gamma_3}$ (Figure 3.27), $Prog_{3,\Gamma_3}$ (Figure 3.28) and $Prog_{4,\Gamma_3}$ (Figure 3.29) that behave respectively like the grammar components G_1, G_2, G_3 and G_4 rewriting the corresponding sentential form with the rewriting rules $P_1 = \{S_1 \rightarrow S_1\text{'}\}$, $P_2 = \{S_2 \rightarrow S_2\text{'}, X \rightarrow c\}$, $P_3 = \{S_3 \rightarrow S_3\text{'}, X \rightarrow c\}$, and $P_4 = \{S_1 \rightarrow aS_4, S_4 \rightarrow bS_4, S_4 \rightarrow X\}$.

The global correctness of the specification below is guaranteed by two things. First by the definition of functions Δ and δ as specified in the definition of CCPC Grammar systems introduced in section 5. Second by the declaration in multiprogram $MProgram_{\Gamma_3}$ (Figure 3.25) of variables R_1, R_2, R_3, R_4 for the respective filters found in the analysis presented in (I) for the grammar components G_1, G_2, G_3, G_4 in Γ_3, such that $R_1 = \{a, b\}^*c$, $R_2 = \{a, b\}^*X$, $R_3 = \{a, b\}^*X$ and $R_4 = \lambda$.

```
$\{[(w_1, w_2, w_3, w_4) = (S'_1, S'_2, S'_3, xX) \vee (w_1, w_2, w_3, w_4) = (S'_1, xc, xc, yX)] \wedge x, y \in \{a, b\}^* \wedge X \in N\}$
$i := 1;$
$do\ i \leq 3 \rightarrow$
        $if\quad \Delta(i) \neq \lambda \rightarrow w_i := \Delta(i);$
            $\mid \Delta(i) = \lambda \wedge \delta(i) = \lambda \rightarrow skip;$
            $\mid \Delta(i) = \lambda \wedge \delta(i) \neq \lambda \rightarrow w_i := S_i;$
  $fi;$
  $i := i + 1;$
$od;$
$\{[(w_1, w_2, w_3, w_4) = (S'_1, xX, xX, S_4) \vee (w_1, w_2, w_3, w_4) = (xcxc, yX, yX, S_4)] \wedge x, y \in \{a, b\}^* \wedge X \in N\}$
```

Because the selection of rewriting rules in programs $Prog_{1,\Gamma_3}$, $Prog_{2,\Gamma_3}$ and $Prog_{3,\Gamma_3}$ is simulated by an indeterministic alternative command *if*, then for all $w \in \{xcxc \mid x \in \{a, b\}^*\}$ multiprogram $MProgr_{\Gamma_3}$ computes w.

(III) As explained in proposition 3 the Grammar system $\Gamma_3 \in CCPC_4(CF)$ is equivalent to the multiprogram $MProgram_{\Gamma_3}$ (Figure 3.25) with programs $Prog_{1,\Gamma_3}$ (Figure 3.26), $Prog_{2,\Gamma_3}$ (Figure 3.27), $Prog_{3,\Gamma_3}$ (Figure 3.28) and $Prog_{4,\Gamma_3}$ (Figure 3.29). Γ_3 is defined in this way:

$$\begin{aligned}\Gamma_3 \quad = \quad & (\{S_1, S_2, S_3, S_4, S_1\text{'}, S_2\text{'}, S_3\text{'}, S_4\text{'}, X\}, \{a, b, c\},\\ & (S_1, P_1, R_1), (S_2, P_2, R_2), (S_3, P_3, R_3), (S_4, P_4, R_4))\end{aligned}$$

where:

$P_1 = \{S_1 \rightarrow S_1\text{'}\}$, $R_1 = \{a, b\}^* c$,

$P_2 = \{S_2 \rightarrow S_2\text{'}, X \rightarrow c\}$, $R_2 = \{a, b\}^* X$,

$P_3 = \{S_3 \rightarrow S_3\text{'}, X \rightarrow c\}$, $R_3 = \{a, b\}^* X$,

$P_4 = \{S_1 \rightarrow aS_4, S_4 \rightarrow bS_4, S_4 \rightarrow X\}$, $R_4 = \lambda$. ■

```
$MProg_{\Gamma_3}$ :  $w_1, w_2, w_3, w_4 : String$;
            $T, N : Set\ of\ Character$;
            $R_1, R_2, R_3, R_4 : String$;
            $i : Integer$;

            $N := \{S_1, S_2, S_3, S_4, S'_1, S'_2, S'_3, S'_4, X\}$;
            $T := \{a, b, c\}$;
            $R_1 := \{a, b\}^* c$;
            $R_2 := \{a, b\}^* X$;
            $R_3 := \{a, b\}^* X$;
            $R_4 := \lambda$;
            $w_1, w_2, w_3, w_4 := S_1, S_2, S_3, S_4$;
            $\{(w_1, w_2, w_3, w_4) = (S_1, S_2, S_3, S_4)\}$
            $do\ contains(w_1, N) \rightarrow$
                $\{(w_1, w_2, w_3, w_4) = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)\}$
                    $q := 0$;
                    $Prog_{1,\Gamma_3} \parallel Prog_{2,\Gamma_3} \parallel Prog_{3,\Gamma_3} \parallel Prog_{4,\Gamma_3} \parallel$
                $\{(w_1, w_2, w_3, w_4) = (\beta_1, \beta_2, \beta_3, \beta_n) \wedge (\forall 1 \le i \le n : w_i = \beta_i \wedge \alpha_i \overset{*}{\underset{P_i}{\Longrightarrow}} \beta_i)\}$
                  $i := 1$;
                  $do\ i \le 3 \rightarrow$
                        $if$          $\Delta(i) \ne \lambda \rightarrow w_i := \Delta(i)$;
                                    $\mid \Delta(i) = \lambda \wedge \delta(i) = \lambda \rightarrow skip$;
                                    $\mid \Delta(i) = \lambda \wedge \delta(i) \ne \lambda \rightarrow w_i := S_i$;
                        $fi$;
                        $i := i + 1$;
                  $od$;
                $\{(w_1, w_2, w_3, w_4) = (\pi_1, ..., \pi_n) \wedge (\beta_1, ..., \beta_n) \vdash (\pi_1, ..., \pi_n)\}$
            $od$;
            $\{w_1 \in L(\Gamma_3)\}$
            $Print(w_1)$
```

Figure 3.25: Multiprogram for Γ_3

```
$Prog_{1,\Gamma_3}$ :   $flag : Boolean;$

        $\langle$  $\{w_1 = \alpha\}$
             $flag := true;$
             $do\ flag \rightarrow$
                 $\{Inv : w_1 = \beta \wedge \alpha \overset{*}{\underset{P_1}{\Longrightarrow}} \beta \wedge flag\}$
                         $if$   $w_1 = \gamma S_1 \delta$   $\rightarrow w_1 := \gamma S'_1 \delta;$
                              $|\ true$          $\rightarrow flag := false;$
                         $fi;$
             $od;$
             $\{w_1 = \beta \wedge \alpha \overset{*}{\underset{P_1}{\Longrightarrow}} \beta \wedge \neg flag\}$ $\rangle$
```

Figure 3.26: Program for G_1 in Γ_3

```
$Prog_{2,\Gamma_3}$ :   $flag : Boolean;$

        $\langle$  $\{w_2 = \alpha\}$
             $flag := true;$
             $do\ flag \rightarrow$
                 $\{Inv : w_2 = \beta \wedge \alpha \overset{*}{\underset{P_2}{\Longrightarrow}} \beta \wedge flag\}$
                         $if$   $w_2 = \gamma S_2 \delta$   $\rightarrow w_2 := \gamma S'_2 \delta;$
                              $|\ true$          $\rightarrow flag := false;$
                         $fi;$
             $od;$
             $\{w_2 = \beta \wedge \alpha \overset{*}{\underset{P_2}{\Longrightarrow}} \beta \wedge \neg flag\}$ $\rangle$
```

Figure 3.27: Program for G_2 in Γ_3

```
$Prog_{3,\Gamma_3}$ :   $flag : Boolean;$

        $\langle$  $\{w_3 = \alpha\}$
           $flag := true;$
           $do\ flag \rightarrow$
              $\{Inv : w_3 = \beta \wedge \alpha \overset{*}{\underset{P_3}{\Longrightarrow}} \beta \wedge flag\}$
                    $if$   $w_3 = \gamma S_3\delta$   $\rightarrow w_3 := \gamma S'_3\delta;$
                         $\mid w_3 = \gamma X\delta$   $\rightarrow w_3 := \gamma c\delta;$
                         $\mid true$           $\rightarrow flag := false;$
                    $fi;$
           $od;$
           $\{w_3 = \beta \wedge \alpha \overset{*}{\underset{P_3}{\Longrightarrow}} \beta \wedge \neg flag\}$ $\rangle$
```

Figure 3.28: Program for G_3 in Γ_3

```
$Prog_{4,\Gamma_3}$ :   $flag : Boolean;$

        $\langle$  $\{w_4 = \alpha\}$
           $flag := true;$
           $do\ flag \rightarrow$
              $\{Inv : w_4 = \beta \wedge \alpha \overset{*}{\underset{P_4}{\Longrightarrow}} \beta \wedge flag\}$
                    $if$   $w_4 = \gamma S_1\delta$   $\rightarrow w_4 := \gamma a S'_4\delta;$
                         $\mid w_4 = \gamma S_4\delta$   $\rightarrow w_4 := \gamma b S_4\delta;$
                         $\mid w_4 = \gamma S_4\delta$   $\rightarrow w_4 := \gamma X\delta;$
                         $\mid true$           $\rightarrow flag := false;$
                    $fi;$
           $od;$
           $\{w_4 = \beta \wedge \alpha \overset{*}{\underset{P_4}{\Longrightarrow}} \beta \wedge \neg flag\}$ $\rangle$
```

Figure 3.29: Program for G_4 in Γ_3

In Grammar System Theory important efforts have been dedicated to finding Grammar systems with the fewest possible number of grammars and more restricted productions, to show how distribution and communication can make simple components very powerful when they work together. For PC Grammar systems some measures of complexity based on the number of communications between grammars have been presented and studied in [HKK94] and [Par93]. Apart from this the most investigated complexity measure has been the number of grammars that a Grammar system consist on, which is clearly a descriptional complexity measure. So a very important matter has been forgotten: the efficient use of time. The opposite happened in the programming area (see [Fos95]), where an important part of the research has been focused on looking for techniques to parallelize algorithms and to help programmers to design more efficient concurrent algorithms.

We state that although there are no recipes to follow, in some cases we can construct efficient Grammar systems following some of the methodical approaches developed in the programming framework that maximize the range of options considered and that provide mechanisms for evaluating alternatives.

For example if we calculate the time that the Grammar system Γ_3 defined above spends to generate a string $xcxc$ with $x \in \{a,b\}^*$ and $\mid x \mid = n$, it is of $O(n)$ in the best case. If we want to improve the efficiency of Γ_3 in terms of time spent to produce a string, we can try to apply some of the strategies developed in the programming framework to design parallel algorithms. For this example we can apply the so-called *functional decomposition.* Functional decomposition is a strategy for partitioning, used for the design of concurrent algorithms. This strategy focuses on the computation to expose opportunities for parallel execution. Hence, the idea is to define a large number of small tasks in order to yield a fine-grained decomposition of a problem. Formally:

Definition 22 *(Functional decomposition) For a multivariate function $y = f(x_1, ..., x_n)$ functional decomposition refers to the process of identifying a set of functions $\{g_1, g_2, ..., g_m\}$ and a function ϕ such that: $f(x_1, ..., x_n) = \phi(g_1((x_1, ..., x_n)), ..., g_m(x_1, ..., x_n))$*

We can apply the functional decomposition strategy over Γ_3 to generate another Grammar system Γ_4 that solves this problem in less time as we prove below:

Theorem 8 $\{xcxc \mid x \in \{a,b\}^*\} \in CCPC(CF)$.

Proof. We focus on the computation of the string $x \in \{a,b\}^*$ and we discover that this task can be done by an arbitrary number $m > 1$ of grammars working simultaneously instead of only one grammar. In this way we can reduce the time to $O(n/m)$, in the best case.

For defining Γ_4 we propose this refinement of **Subproblem 1**:

$\{w_1 = S_1 \wedge w_2 = S_2 \wedge w_3 = S_3 \wedge w_4 = S_4\}$
Subproblem 1: *(Rewrite)*$^{n+1}$, $n \geq 0$
$Prog_4$ generates xX, $x \in \{a,b\}^*, \mid x \mid = n$.
$\{w_1 = S_1' \wedge w_2 = S_2' \wedge w_3 = S_3' \wedge w_4 = xX \wedge x \in \{a,b\}^* \wedge \mid x \mid = n\}$

To improve efficiency we propose another refinement of the same subproblem:

$\{w_1 = S_1 \wedge w_2 = S_2 \wedge w_3 = S_3 \wedge w_4 = S_4 \wedge ... \wedge w_{i+4} = S_{i+4} \wedge ...$
$... \wedge w_{m+3} = S_{m+3} \wedge w_{m+4} = S_{m+4} \wedge 1 \leq i \leq m-1 \wedge 1 \leq m\}$

Subproblem 1
$$\left\{\begin{array}{l}
(Rewrite)^{t+1},\ t \geq 0 \\
Prog_{i+4}, ..., Prog_{m+3} \text{ generates } x_i, ..., x_{m-1} \in \{a, b\}^*, \\
1 \leq i \leq m-1, 1 \leq m \text{ and } Prog_{m+4} \text{ generates } x_m Y, \\
x_m \in \{a, b\}^* \\
\{w_1 = S_1\text{'} \wedge w_2 = S_2\text{'} \wedge w_3 = S_3\text{'} \wedge w_4 = S_4\text{'} \wedge ... \\
... \wedge w_{i+4} = x_i \wedge ... \wedge w_{m+3} = x_{m-1} \wedge w_{m+4} = x_m Y \wedge \\
\wedge 1 \leq i \leq m-1 \wedge 1 \leq m \wedge x_1, ..., x_m \in \{a, b\}^* \wedge Y \in N\} \\
Communication \\
Prog_4 \text{ receives the } x_1, ..., x_{m-1} \in \{a, b\}^* \text{ produced by} \\
Prog_5, ..., Prog_{m+3} \text{ followed by } x_m Y, \\
x_m \in \{a, b\}^* \text{ produced by } Prog_{m+4} \\
\{w_1 = S_1\text{'} \wedge w_2 = S_2\text{'} \wedge w_3 = S_3\text{'} \wedge w_4 = x_1 ... x_m Y \wedge \\
\wedge ... \wedge w_{i+4} = S_{i+4} \wedge ... \wedge w_{m+3} = S_{m+3} \wedge w_{m+4} = S_{m+4} \wedge \\
\wedge 1 \leq i \leq m-1 \wedge 1 \leq m \wedge x_1 ... x_m \in \{a, b\}^* \wedge Y \in N\} \\
(Rewrite)^{s+1},\ s \geq 0 \\
Prog_4 \text{ replaces } Y \text{ by } X.
\end{array}\right.$$

$$\begin{array}{l}
\{w_1 = S_1\text{'} \wedge w_2 = S_2\text{'} \wedge w_3 = S_3\text{'} \wedge w_4 = x_1 ... x_m X \wedge ... \wedge w_{i+4} = y_i \wedge \\
\wedge ... \wedge w_{m+3} = y_{m-1} \wedge w_{m+4} = y_m Y \wedge 1 \leq i \leq m-1 \wedge 1 \leq m \wedge \\
\wedge x_1 ... x_m \in \{a, b\}^* \wedge y_1, ..., y_m \in \{a, b\}^* \wedge Y \in N\}
\end{array}$$

The rest is analogous to the analysis we made for Γ_3, and according to our previous analysis we get $\Gamma_4 \in CCPC_{m+4}(CF)$, $m \geq 1$ defined in this way:

$$\begin{aligned}
\Gamma_4 \;=\; & (\{S_1, S_2, S_3, ..., S_{m+4}, S_1\text{'}, S_2\text{'}, S_3\text{'}, ..., S_{m+4}\text{'}, X, Y\}, \{a, b, c\}, \\
& (S_1, P_1, R_1), (S_2, P_2, R_2), ..., (S_{m+4}, P_{m+4}, R_{m+4}))
\end{aligned}$$

where:

$P_1 = \{S_1 \rightarrow S_1\text{'}\}, R_1 = \{a, b\}^* c,$

$P_2 = \{S_2 \rightarrow S_2\text{'}, X \rightarrow c\}, R_2 = \{a, b\}^* X,$

$P_3 = \{S_3 \rightarrow S_3\text{'}, X \rightarrow c\}, R_3 = \{a, b\}^* X,$

$P_4 = \{S_4 \rightarrow S_4\text{'}, Y \rightarrow X\}, R_4 = \{a, b\}^* Y \cup \{a, b\}^*,$

$P_{i+4} = \{S_{i+4} \rightarrow aS_{i+4}, S_{i+4} \rightarrow bS_{i+4}, S_{i+4} \rightarrow a, S_{i+4} \rightarrow b, S_{i+4} \rightarrow \lambda\}, R_{i+4} = \emptyset, 1 \leq i \leq m-1$

$P_{m+4} = \{S_{m+4} \rightarrow aS_{m+4}, S_{m+4} \rightarrow bS_{m+4}, S_{m+4} \rightarrow Y\}, R_{m+4} = \emptyset, 1 \leq m$ ■

We improve efficiency, because Γ_4 generates strings $xcxc$ with $x \in \{a, b\}^*$ and $| x |= n$ in $O(n/m)$, in the best case. In more general terms we show with theorem 8 that we can use strategies available in the programming framework to design Grammar systems that derive strings in less time.

3.3.3 How can programming benefit from Grammar systems?

If for example we want to prove in the Grammar System Theory that there is no Grammar system with n components with a certain protocol of communication that generates a language L, we use analysis by cases and induction strategies. If we translate this problem to the programming framework, we have to prove that it is not possible to find a multiprogram $\mathcal{P}$ with n programs running concurrently, communicating following the same protocol, and that is correct with respect to this specification:

$$\{(w_1 = S_1) \wedge (w_2 = S_2) \wedge ... \wedge (w_n = S_n) \wedge n \geq 1\}\ \mathcal{P}\ \{w_1 \in L\}$$

But in the programming framework we have no strategies for reasoning in the negative way. The only strategies available in this framework are: *verification* that, given a multiprogram consists of proving its correctness with respect to a specification (for example the second proof of theorem 6), and the *constructive approach* that we have exemplified with theorems 6, 7 and 8 that consists of constructing simultaneously a program and the proof of its correctness with respect to a specification. Both strategies are useful for obtaining positive results.

The lack of strategies to prove these kind of negative results in the programming framework suggests the possibility of translating them to the Grammar system framework and using the tools available there solve them. We exemplify this approach with the language L_{cd} already introduced. We want to prove that there is no Grammar system with two regular components that can generate L_{cd}. In other words we want to prove that the solution we proposed, $L_{cd} \in NPC_3(Reg)$, is the most economical one with respect to the number of components. If we translate this problem into the programming framework, we have to prove that it is not possible to find a multiprogram $\mathcal{P}$ with two programs $Prog_1$ and $Prog_2$ running concurrently, modifying w_1 and w_2 in a regular way, such that $\{(w_1 = S_1) \wedge (w_2 = S_2)\}\ \mathcal{P}\ \{w_1 \in \{a^n b^m c^n d^m \mid n, m \geq\}\}$ is correct. We solve this problem with the tools available in the Grammar system framework, namely analysis by cases.

Theorem 9 *$L_{cd} \notin X_2(Reg)$, for $X \in \{PC, CPC, NPC, NCPC\}$.*

Proof. Since $PC_2(Reg)$ (hence $CPC_2(Reg)$, too), contains context free languages only [CVJJP94], it suffices to prove that L_{cd} cannot be in $NPC_2(Reg)$. Assume that $L_{cd} = L(\Gamma)$ for some non-returning non-centralized Grammar system with two regular components Γ. Take $w = a^n b^m c^n d^m$ with arbitrarily large n, m. There exist two nonterminals A_1, A_2 such that in the process of generating w the following holds:

$$\begin{aligned} (S_1, S_2) &\Longrightarrow^* (x_1 A_1, x_2 A_2), x_1, x_2 \in \{a, b, c, d\}^*, \\ (A_1, A_2) &\Longrightarrow^{k_1} (u A_1, v A_2), u, v \in \{a, b, c, d\}^*, uv \neq \lambda \end{aligned}$$

for some $k_1 \geq 1$. Here $\Longrightarrow^p$ denotes a derivation of length p where the communication steps are also counted. Let (A_1, A_2) be the first pair of such nonterminals met in the process with this property. By the choice of w we infer that $u \in a^+$ and $v \in c^+$. First we note that both u and v, if non-empty, are formed by one letter only. Second, if one is empty, then the other can be "pumped" arbitrarily many times which leads to a parasitic word. Third, by the same argument, all the other choices, except $u \in a^+$ and $v \in c^+$, lead to words not in L_{cd}. Since the subword of w formed by b is arbitrarily long there is a pair of nonterminals (B_1, B_2) such that the derivation continues as follows:

$$\begin{aligned}&(S_1, S_2) \Longrightarrow^* (x_1A_1, x_2A_2) \Longrightarrow^{rk_1} (x_1u^rA_1, x_2v^rA_2) \Longrightarrow^k \\ &(x_1u^ry_1B_1, x_2v^ry_2B_2) \Longrightarrow^{pk_2} (x_1u^ry_1s^pB_1, x_2v^ry_2t^pB_2) \Longrightarrow^* (w, \alpha)\end{aligned}$$

for some terminal words $y_1, y_2, s, t,\ s \in b^+$, positive integers r, p, k, k_2, and word α. Moreover, no communication step appears in the subderivation

$$(x_1u^rA_1, x_2v^rA_2) \Longrightarrow^* (x_1u^ry_1B_1, x_2v^ry_2B_2) \Longrightarrow^{pk_2} (x_1u^ry_1s^pB_1, x_2v^ry_2t^pB_2)$$

otherwise either an insufficient number of b's is generated between the two subwords formed by a and c or the generated word contains the subword cb which is contradictory.

Therefore, the following derivation is also possible in Γ:

$$\begin{aligned}&(S_1, S_2) \Longrightarrow^* (x_1A_1, x_2A_2) \Longrightarrow^{rk_1} (x_1u^rA_1, x_2v^rA_2) \Longrightarrow^{k_1k_2+k} \\ &(x_1u^{r+k_2}y_1B_1, x_2v^ry_2t^{k_1}B_2) \Longrightarrow^{pk_2} (x_1u^{r+k_2}y_1s^pB_1, x_2v^ry_2t^{p+k_1}B_2) \\ &\Longrightarrow^* (w', \alpha'),\end{aligned}$$

for terminal word w' and word α'. However, w' cannot be in L_{cd}, which concludes the proof. ■

As we have pointed out before, in the Grammar system framework an important part of the research is concentrated on reducing the total number of components in the grammatical system to prove the power of communication. The opposite happens in the programming framework where programmers try to partition programs in as many tasks as possible to improve efficiency in time, looking for strategies to parallelize programs. So it looks like research is directed in a different direction. But there are some results of Grammar System Theory that can benefit the concurrent Programming framework, like these theorems from [CVJJP94] that allow a transformation of a Grammar system of at least m grammars into a Grammar system of at least n grammars that generates the same language:

$$CF = CD_{1,*}(t) = CD_{2,*}(t) \subset CD_{3,*}(t) \text{ and}$$

$$CD_{3,*}(t) = CD_{*,*}(t) = ET0L$$

There are another theorems of this type in Grammar System Theory that translated to the programming framework speak about the number of programs needed to generate a certain language (refer to [CVJJP94]). This can be considered a contribution from Grammar System Theory to programming framework, where there are not results concerning the number of programs needed to solve a problem. It would be very interesting for the design of concurrent programs if some of these transformations can also consider efficiency. If there were some results about how to transform a program $\mathcal{P}$ that has m multiprograms running concurrently into a program with n multiprograms that solve the same problem more efficiently, this could be a great contribution to the Multiprogramming Theory.

Chapter 4

Using Grammar systems for specifying dialogue protocols

4.1 Introduction

In this chapter we introduce a formal framework that we call *Conversational Enlarged Reproductive Eco-Grammar* (*ConvEREG*) system for modeling goal-oriented dialogue protocols. *ConvEREG* systems are based on a Grammar system variant that we introduce in section 4.2 as *Enlarged Reproductive Eco-Grammar* (*EREG*) systems. In *ConvEREG* the grammatical components are interpreted as speakers whose participation in the conversation is described by protocols, strings in a process calculus called *Multi Agent Protocol (MAP^a)* language that we defined in [GW06b] [GW06c] [GW06a] and we present in section 4.3.1. Our approach can be seen as a continuation of similar models presented in [CVJLMV99], [JL00] and [AJR01].

Dialogue protocols specify complex concurrent and asynchronous patterns of communication of social norms in a society. Protocols can be defined by means of finite automata [Vas04, ERS$^+$01], high level Petri Nets [HK98][MW97][PC96], diagrams provided by the Unified Modeling Language [Woo99] [OPB00] [WCW01] [HWW01], logic [Woo99] [GK94], and process descriptions [Rob04] [Wal04c]. A society protocol is examined in-advance by an agent in order to decide if it joins or not the corresponding society. The protocol also acts as a guide for the agents to follow once they are operating within the society. Based on the *Multi Agent Protocol* (*MAP*) language [Wal04c] we define the MAP^a calculus for specifying more flexible, dynamic and close-to-human process-based protocols of communication and coordination between agents. In [GW06b] [GW06c] [GW06a] we explain why the MAP^a calculus represents an improvement with respect to the MAP language.

The *ConvEREG* systems that we define in section 4.3 allow the specification of protocols as dialogue structures because they focus only on those agent's observable behaviors that are part of the interaction, abstracting from the implementation details. Dialogue protocols specified with *ConvEREG* systems do not involve natural language processing, they correspond to information state theory, they are linguistically well founded, generic and highly expressive. A preliminary definition of the framework that we present here has been published in [BEGJL06] [EGJL06] [EGJL07]. Compared to other grammatically based frameworks for the definition of conversations, the novelty of our approach is given by defining the speakers' (grammars) behavior in terms of string process descriptions (protocols) in the MAP^a calculus, instead of defining it in terms of rewriting rules. While rewriting rules remain fixed, strings can be modified during the course of the conversation providing dialogues with great dynamism and flexibility. Also string descriptions provide a more intuitive way to understand and describe the behavior of the agent than rewriting rules.

In section 4.3.3 we exemplify the use of *ConvEREG* systems with the definition of a dialogue protocol. We show that one of the benefits of specifying dialogues in a formal framework is the formal proof of dialogue properties. In section 4.4 we explain why our framework is an improvement with respect to the previous grammatically based attempts to model dialogue. We devote section 4.5 to summarize the main characteristics of our framework. We prove that an *Extended ConvEREG* (*EConvEREG*) system with an arbitrary number of $n \geq 1$ active agents has the same expressive power as Turing Machines.

In section 4.7 we introduce a type of *ConvEREG* system that we call *Conditional Eco-Grammar* (*CondEG*) system. *CondEG* systems correspond to a variant of *Eco-Grammar* system where the private knowledge bases of the agents are restricted to two finite sets of conditional rules that agents use to perform computable decision procedures. According to the first set of rules an agent decides its participation in the dialogue by checking the presence and\or absence of information in its mental state. The agent uses the second set of rules to decide how to change its mental state according to the presence and\or absence of information in the dialogue context. In section 4.7.2 we prove some formal properties for *Extended Conditional Eco-Grammar (ECondEG)* systems. With theorem 11 we connect the family of classes of languages generated by *ECondEG* systems with the family of classes of languages generated by *Extended Conditional Tabled Eco-Grammar (ECTEG)* systems, proving that $ECTEG_n(i, j, c) \subseteq ECondEG_1(i, j, c)$ for all $i, j \geq 0$, $n \geq 1$ and $c \in \{b, s\}$. With theorem 12 we prove that *ECondEG* systems with non erasing rules have the same expressive power as CS grammars.

In section 4.8 we introduce a type of *ConvEG* system that we call *regularly Controlled Reproductive Eco-Grammar (rCREG)* system. *rCREG* systems are a variant of Reproductive Eco-Grammar system where all the rewriting rules R_i used by $1 \leq i \leq n$ agents are λ-free, mapping ψ_i returns an arbitrary production from R_i, and a turn talking policy given by a regular set is attached to determine the order that speakers must respect to participate in the dialogue. In section 4.8.3 we introduce some formal properties for *Extended regularly Controlled Reproductive Eco-Grammar (ErCREG)* systems. We prove with theorem 13 that *ErCREG* systems have the same expressive power as *rC regularly Controlled* grammars with λ-free CF productions [GS68] [DPS97]. Where *rC* is a mildly CS class in the Chomsky hierarchy. Therefore for *ErCREG* systems the problems of membership, emptiness and finiteness are decidable, with the two last problems being NP-hard.

In this chapter we propose a hierarchy of formal frameworks for the specification of dialogue protocols based on Grammar systems. Apart from investigating the expressiveness of the frameworks, we analyze their suitability for the specification of dialogues. In the next chapter we enlarge this hierarchy proposing another frameworks for the definition of dialogue protocols based on finite state transition systems. The overall intention of this and the next chapter is to contribute to Dialogue Theory, mainly from the perspective of Formal Language Theory, to the formal study of frameworks for the specification of dialogues

4.2 Enlarged Reproductive Eco-Grammar systems

In definition 23 we introduce the Grammar system variant that we use for the specification of our framework, that we call *Enlarged Reproductive Eco-Grammar* (EREG) system. EREG systems are the result of modifying the definition of *Reproductive Eco-Grammar* (REG) systems introduced in [CVKKP97] and explained in detail in section 2.3.4. We introduce these modifications over REG systems:

1. The set of action rules R_i is a finite set of rules of the form $x \to y$ with $x \in V_E^+$ and $y \in V_E^*$, or $x \in V_i^+$ and $y \in V_i^*$. In this way we extend the capacities of the agents such that they can, apart from modifying the environment state, modify its own state. Then an agent $A_{i,j}$ can select using computable function $\psi_{i,j}$, not necessarily complete, a set of action rules R_1 from R_i to change its current state $w_{i,j}$. Also the agent can select using computable function $\psi_{i,j}$, not necessarily complete, a set of action rules R_2 from R_i for rewriting symbols from the environmental state w_E. The purpose of this change is to model how agents can simultaneously participate in the conversation (change the state of the environment) and change their mental state.

   With regard to the possible conflict between action rules selected by mapping $\psi_{i,j}$ and action rules selected by mapping $\varphi_{i,j}$, the criteria to avoid this problem is the following: rules selected by mapping $\varphi_{i,j}$ are applied over $w_{i,j}$ and then action rules selected by mapping $\psi_{i,j}$.

2. The set of evolution rules P_i is a finite set of rules of the form $x \to y$ with $x \in V_i^+$ and $y \in V_i^*$.

The resulting EREG system definition inherits all the features that according to [CVJLMV99] make Grammar systems appropriate to model conversation: distribution, modularity, parallelism, interaction, coordination and emergent behavior.

Definition 23 *An Enlarged Reproductive Eco-Grammar (EREG) system of degree* n *is a (n+1)-tuple* $\Sigma = (E, \mathcal{A}_1, ..., \mathcal{A}_n)$ *that only differs from a Reproductive Eco-Grammar system in the following:*

- *In the definition of the agent A_i in class $\mathcal{A}_j$, $1 \leq j \leq n$, the set R_i is a finite set of rules of the form $x \to y$ with $x \in V_E^+$ and $y \in V_E^*$ or, $x \in V_i^+$ and $y \in V_i^*$.*
- *In the definition of the agent A_i the evolution rules P_i are given by a finite set of rules of the form $x \to y$ with $x \in V_i^+$ and $y \in V_i^*$.*

In order to describe the dynamic aspects of EREG systems we give below the following definitions:

Definition 24 *(System configuration) A configuration of an EREG system $\Sigma = (E, \mathcal{A}_1, ..., \mathcal{A}_n)$ is a (n+1)-tuple: $\sigma_t = (w_E, W_1, W_2, ..., W_n)$ where $w_E \in V_E^*$ and W_i is a multiset of strings $w_{i,1}, ..., w_{i,k_i}$, where $w_{i,j} \in V_i^+$ for $1 \leq j \leq k_i$, $1 \leq i \leq n$; w_E is the evolution state of the environment at time t and the strings $w_{i,1}, ..., w_{i,k_i}$ correspond to the evolution states of the active agents of the i-th type at time t.*

Definition 25 *(Agent Derivation) Agent state $w_{i,j}$ derives to a new state $w'_{i,j}$ denoted by $w_{i,j} \vdash w'_{i,j}$ if first $w_{i,j} \vdash_1 \beta'$ as the result of the simultaneous application of as many rules as possible from the set of rules selected by mapping $\psi_{i,j}$ from agent $A_{i,j}$. Then $\beta' \vdash_2 w'_{i,j}$ as the result of the simultaneous application of as many rules as possible from the set of rules selected by mapping $\varphi_{i,j}$.*

Definition 26 *(Environment Derivation) Environmental state w_E derives to new state w'_E denoted by $w_E \models w'_E$ iff first $w_E \models_1 \alpha'$ as the result of the simultaneous application of as many rules as possible from the set of the environment rewriting rules selected by mappings $\psi_{i,j}$ from all active agents, and later $\alpha' \models_2 w'_E$ according to the environment's rules.*

Definition 27 *(System derivation) Let $\Sigma = (E, \mathcal{A}_1, ..., \mathcal{A}_n)$ be an EREG system and let $\sigma_t = (w_E, W_1, W_2, ..., W_n)$ and $\sigma'_{t+1} = (w'_E, W'_1, W'_2, ..., W'_n)$ be two configurations of Σ. We say that σ_t is directly changed for (it directly derives) σ'_{t+1}, denoted by $\sigma_t \Longrightarrow_\Sigma \sigma'_{t+1}$, iff w'_E arises from w_E by evolution of the environment affected by all agents in σ_t, and W'_i is the reproduction of the states actually evolving from the states in W_i, $1 \leq i \leq n$. Note that action rules have priority over evolution rules.*

We denote as $\overset{+}{\Rightarrow}$ and $\overset{*}{\Rightarrow}$ the transitive and reflexive closure of $\Longrightarrow$.

Definition 28 *(System language) The language generated by an EREG system Σ with initial configuration σ_0 is defined as*

$$L_E(\Sigma, \sigma_0) = \left\{ \begin{array}{c} w_E \in V_E^* \mid \sigma_j = (w_E, W_1, ..., W_n), \\ \sigma_0 \Longrightarrow_\Sigma \sigma_1 \Longrightarrow_\Sigma .... \Longrightarrow_\Sigma \sigma_j, j \geq 0 \end{array} \right\}$$

Definition 29 *(Final configuration) Given $\Sigma \in EREG$ with $n \geq 1$ populations, we say that a system configuration $(w_E, W_1, ..., W_n)$ is final in Σ iff*

$$\neg(\exists(w'_E, W'_1, ..., W'_n) : (w_E, W_1, ..., W_n) \Rightarrow^*_\Sigma (w'_E, W'_1, ..., W'_n) \wedge (w_E, W_1, ..., W_n) \neq (w'_E, W'_1, ..., W'_n)).$$

We denote by $EREG_n$, $n \geq 1$, the family of languages $L_E(\Sigma, \sigma_0)$ where Σ is an *EREG* system with degree at most n and initial configuration σ_0.

4.3 *ConvEREG* systems

We introduce first the *Multi-Agent Protocol* (MAP^a) calculus that we use for describing the behavior of speakers in the *ConvEREG* systems.

4.3.1 The MAP^a Language

The MAP^a language is a formal process calculus for the definition of goal-oriented dialogues. In MAP^a the processes are called protocols and they express social interactions between groups of agents. The key concepts in MAP^a are *scenes*, *roles* and *protocols* that we now describe.

A *scene* is conceptually a bounded space in which a group of agents interact on a single task. We assume that a scene is initialized with a set of active agents which start the dialogue and is concluded when all the agents have concluded their participation in the scene. A scene definition comprises a set of roles and protocols required to accomplish a task, a set of locutions and a set of knowledge to be shared and understood by all the agents playing a role within the scene, and a way to interchange common knowledge. An example scene is shown in figure 4.3 where a buyer agent interacts with a set of sellers with the purpose of buying some items.

The concept of an agent *role* is also central to the definition of our protocols. Agents entering a scene assume an initial role, though this role may change during the scene. By adopting a specific role an agent compromises to perform the protocol associated with that role. This requires the agent to know how to perform the decision operations involved in the role protocol. On the other hand adopting a role allows the agent to have access to the role knowledge. For our example in figure 4.3 agents with the roles of *buyer (B)* and *seller (S)* are defined. When adopting the role of buyer an agent learns the market situation and some commercial strategies that guide him in taking decisions. Roles are defined as a hierarchy, where more specialized roles appear further down in the graph of roles and inherit knowledge and decision procedures from upper roles. For example, an agent may initially assume the role *buyer* but may change to the more specialized role *car buyer* during a scene. This will allow the buyer to obtain knowledge and perform actions related specifically to the car market, which a generic buyer does not need to know.

For each role in a scene, a *protocol* is defined that describes the sequence of operations that an agent performing that role needs to follow in order to cooperate with other agents and not break the conventions. Therefore protocols are the way used to express and guaranty the satisfaction of societal norms and rules. It is important to note that a protocol only contains operations that are specific to the mechanisms of communication and coordination between agents. This makes it straightforward to understand the operation of the protocol without extraneous details, and makes it possible to verify the protocols using automated means, e.g. model checking [Wal04a] [Wal04b]. All the other agent facilities, e.g. the reasoning processes, are encapsulated by *decision procedures* that are external to the protocol. In effect, the decision procedures provide an interface between the communicative and the rational process of the agent. In MAP^a we distinguish two levels of decision procedures: those private to the agent, and those shared between all the agents in the same role.

Every message has an associated *locution* that is used to indicate the type of the message and parameters. For convenience, we do not assign any fixed semantics to these locutions. However, individual agents can agree on a semantics for a particular scene.

The final concept in MAP^a is the representation of *knowledge*. The language allows to define knowledge as sets of axioms at the scene level, the role level, and the level of a particular agent. In this way we can clearly establish differences between the knowledge. Scene and role knowledge is common knowledge that can be accessed by all agents in the scene or role, respectively. Scene knowledge and role knowledge cannot be modified. In contrast, the private knowledge of the agent cannot be accessed externally and can be freely modified by the agent to whom it corresponds.

We show the formal syntax of MAP^a in figure 4.1 in Backus Normal Form. Superscripts are used to indicate a set, e.g. $P^{(i)}$ is a set with elements P of size i.

According to figure 4.1 a MAP^a scene comprises a set of *initial agents* $AI^{(h)}$, a *role hierarchy* $R^{(i)}$, a set of protocols $P^{(i)}$ that are parameterized on these roles, a set of axioms $K^{(b)}$ that is the *common knowledge* in the scene and a set of locutions $M^{(k)}$ which defines the *dialogic structure* (i.e. all the allowed locutions) for the scene.

$$
\begin{array}{llll}
s \in Scene & ::= & \langle AI^{(h)}, R^{(i)},\ P^{(i)},\ K^{(b)},\ M^{(k)} \rangle & \text{(Scene)} \\
AI \in AgentInvocation & ::= & \mathbf{agent}(\phi_1, \phi_2,\ Proc^{(v)},\ K^{(p)}, \phi_3^{(w)}) & \text{(Agent Invocation)} \\
R \in Role & ::= & \langle \phi_1,\ Proc^{(l)},\ K^{(m)},\ R^{(w)} \rangle & \text{(Role)} \\
P \in Protocol & ::= & \mathbf{agent}(\phi_1, \phi_2, \phi_3^{(f)}) :: op & \text{(Protocol)} \\
K & ::= & Axiom & \text{(Knowledge)} \\
Proc \in Procedure & ::= & type :: p((\phi_1, type)^{(g)}) & \text{(Procedure)} \\
M \in Speech\ Act & ::= & \rho((\phi_1, type)^{(h)}) & \text{(Locution)} \\
op \in Operation & ::= & \phi_1 & \text{(Operational variable)} \\
 & \mid & op\ \mathbf{then}\ op & \text{(Sequence)} \\
 & \mid & op\ \mathbf{or}\ op & \text{(Choice)} \\
 & \mid & (op) & \text{(Precedence)} \\
 & \mid & \alpha & \text{(Action)} \\
\alpha \in Action & ::= & \mathbf{null} & \text{(No Action)} \\
 & \mid & \phi_1 = p(\phi_2^{(g)}) & \text{(Decision)} \\
 & \mid & \rho(\phi_1^{(x)}) \Longleftarrow \mathbf{agent}(\phi_2, \phi_3) & \text{(Receive)} \\
 & \mid & \rho(\phi_1^{(y)}) \Longrightarrow \mathbf{agent}(\phi_2, \phi_3) & \text{(Send)} \\
 & \mid & AI & \text{(Agent invocation)} \\
\phi_i \in Term & ::= & constant \mid variable \mid \_ & \text{(Term)}
\end{array}
$$

Figure 4.1: MAP^a language syntax

The role hierarchy is defined as a set of role definitions. Each role has a unique identifier, a set of decision procedures which are shared within the role, a set of axioms which are common to the role and, finally, a set of upper role identifiers, which appear above the role in the hierarchy. This set will be empty for a top-level role.

A protocol is defined by a set of parameters and a body op. The parameters comprise a unique identifier for the protocol, a role, and other parameters. These parameters can be a set of procedures and axioms which are private to the agent.

As previously noted, knowledge is represented by axioms within a protocol. These axioms represent facts which are believed to be true. The reasoning over this knowledge is performed by decision procedures which are formally defined giving the type of their incoming an outgoing parameters. Decision procedures are external to the protocol and have full access to the scene knowledge, the knowledge of the played role and upper roles, and the agent private knowledge. The result of the procedure evaluation is bound to variables.

The core of the protocols are constructed from operations which control the flow of execution, and actions which have side-effects and can fail. The operations can be defined by a variable, by a sequence *then* or by a non deterministic choice *or*. Parenthesis can be used to enforce precedence of operations. It is possible to introduce operational variables during design time, which are replaced during run-time by operations. This feature provides MAP^a protocols with dynamism and with the possibility of defining protocols where part of the behavior is not known beforehand but depends on agents' interaction during the scene execution. With respect to the other operations, they have a standard interpretation. A sequence means that the first operation op_1 is evaluated before the second operation op_2. The non-deterministic choice means that either op_1 or op_2 is evaluated.

The actions an agent can perform are: null action, invocation of decision procedures, receiving and sending messages, and introducing agent invocations. A null action is included because it is convenient for protocol termination. Decision procedures are interfaces between the dialogue protocol and the rational processes of the agent, which allow each agent to modify its own knowledge. The interchange of messages requires the indication of the identifier and the role of the agents that send or receive the message, respectively. It also requires the content or locution $\rho(\phi^{(x)})$. With respect to the action of introducing agent definitions, it is a very powerful mechanism that can be used for creating new agent instances, changing the agent role or keeping the current role but changing parameters for simulating recursive calls.

Here in after we will use the convention of denoting agent identifier values with letters a, b and c, and role identifier values with letters r and l.

4.3.2 Formal Definition

Definition 30 *From an arbitrary MAP^a scene $s = \langle InitAg^{(m)}, Role^{(n)},\ Prot^{(n)},\ SceneK^{(w)},\ DF^{(k)}\rangle$ where all the role and agent private decision procedures are computable functions, we can define $\Sigma_s \in EREG_n$ and we say that $\Sigma_s \in ConvEREG_n$, being Σ_s defined in the following way:*

1. $\Sigma_s = ((V_E, P_E), \mathcal{A}_1, ..., \mathcal{A}_n)$

2. $\tau_{\Sigma_s,0} = (w_{E,0}, W_{1,0}, ..., W_{n,0})$ *where:*

   - $w_{E,0} = SRoles + +SProts + +SScenK + +SDF + +\alpha$, *with*
     $V_E = V_1 \cup V_2$,
     $SRoles,\ SProts,\ SDF,\ SSceneK \in V_1^*$ *are the string representations of* $Role^{(n)},\ Prot^{(n)}$, $DF^{(k)}$ *and* $SceneK^{(w)}$ *respectively,*
     $V_1 \cap V_2 = \emptyset$,
     If $InitAg^{(m)} = \ agent(a_1, r_1, PProc_1^{(s_1)}, PKnow_1^{(t_1)}, Value_1^{(q_1)})...$
     $\qquad agent(a_m, r_m, PProc_m^{(s_m)}, PKnow_m^{(t_m)}, Value_m^{(q_m)})$
     then $\alpha \in V_2^*$ *has the following shape:*
     $$\begin{aligned}\alpha \ = \ & \langle agent(a_1, Sr_1, Scr_1, SPProc_1, SPKnow_1, SVs_1)\rangle_{NA,\_,\_} \\ & \langle\rangle_{NA,r_1,a_1}\langle\rangle_{M,r_1,a_1}\langle\rangle_{AA,r_1,a_1}\cdots \\ & \langle agent(a_m, Sr_m, Scr_m, SPProc_m, SPKnow_m, SVs_m)\rangle_{NA,\_,\_} \\ & \langle\rangle_{NA,r_m,a_m}\langle\rangle_{M,r_m,a_m}\langle\rangle_{AA,r_m,a_m}\end{aligned}$$

     where each string Scr_i, $1 \le i \le m$, is defined as $[\_, \_, \_]$ to indicate that the initial agent instances are not created upon request of any agent.

     Each string $\langle agent(a_i, Sr_i, Scr_i, SPProc_i, SPKnow_i, SValue_i)\rangle_{NA,\_,\_}$ in α corresponds to the string representation of the request to create a new agent instance $agent(a_i, r_i, PProc_i^{(s_i)}, PKnow_i^{(t_i)}, Value_i^{(q_i)})$ from $InitAg^{(m)}$.

     Symbols $\langle\rangle_{NA,r_i,a_i}\langle\rangle_{M,r_i,a_i}\langle\rangle_{AA,r_i,a_i}$ are introduced to be used during the dialogue by the agent with identifier a_i playing role r_i to request the creation of new agent instances, send messages and request the activation of agent instances.

- $W_{j,0} = \{\langle Parent\rangle\}$, *for all* $1 \leq j \leq n$.

3. *Mappings* φ_i, $1 \leq i \leq n$, *are computable functions defined by a set of rules that use the following auxiliary functions:*

$$\begin{aligned}
&getProceds :: Term \to 2^{Procedure}\\
&getProceds(r) = RoleProcs,\ \text{if}\ \langle r, RoleProcs, RoleK, \emptyset\rangle \in Role^{(n)}\\
&getProceds(r) = RoleProcs \cup getProceds(ur^{(1)}) \cup ... \cup getProceds(ur^{(k)}),\\
&\qquad\qquad \text{if}\ \langle r, RoleProcs, RoleK, \{ur^{(1)}, ..., ur^{(k)}\}\rangle \in Role^{(n)}
\end{aligned}$$

$$\begin{aligned}
&getKnowledge :: Term \to 2^{Axiom}\\
&getKnowledge(r) = RoleK,\quad \text{if}\ \langle r, RoleProcs,\ RoleK,\ \emptyset\rangle \in Role^{(n)}\\
&getKnowledge(r) = RoleK \cup getKnowledge(ur^{(1)}) \cup ... \cup getKnowledge(ur^{(k)}),\\
&\qquad\qquad \text{if}\ \langle r, RoleProcs,\ RoleK,\ \{ur^{(1)}, ..., ur^{(k)}\}\rangle \in Role^{(n)}
\end{aligned}$$

Function getProceds retrieves the procedures accessible by a role through a recursive traversal of the hierarchy of roles. The base case applies when there is no upper role, retrieving the role's procedures. When the recursive case occurs the procedures are retrieved from the upper roles. Function getKnowledge behaves like function getProceds but retrieves all the knowledge accessible by a role.

The rules that define mappings φ_i, $1 \leq i \leq n$, *are the following:*

[φ_i.1] *If an agent* $A_{r_i,(a,x)}$ *contains the symbol* $\langle Parent\rangle$ *in* $w_{r_i,(a,x)}$, *it means that the agent playing the role* r_i *with identifier a, that was created at derivation time x, has received the request to introduce new agent instances with role* r_i. *The agent* $A_{r_i,(a,x)}$ *detects this request when mapping* φ_i *finds in* w_E *the presence of g substrings* $\langle agent(SName_y, Sr_j, Scr_y, SPProc_y, SPKnow_y, SValues_y)\rangle_{NA,c_y,k_y}$, $1 \leq y \leq g$ *such that:*

(a) *identifiers* $SNmae_1, ..., SName_g$ *are all different,*

(b) *there are not active agents in* Σ_s *with identifiers* $SNane_1, ..., SName_g$ *playing role* r_i. *Then symbols* $\langle\rangle_{M,r_i,SName_y}$, *for all* $1 \leq y \leq g$ *do not occur in* $w_{E,0}$.

If these conditions are satisfied mapping φ_i *selects the following set of rules to create the requested new agent instances:*

$$\left\{\begin{array}{ll}
\langle Parent\rangle \longrightarrow & \sqcup SName_1\ Scr_1 \downarrow \langle\rangle_M \langle SOp_1\rangle_{OP}\ \langle Active\rangle\ SUTerms_1\ SSk\\
& SAKnow_1\ SAProc_1 \sqcup ...\\
& \sqcup SName_g\ Scr_g \downarrow \langle\rangle_M \langle SOp_g\rangle_{OP}\ \langle Active\rangle\ SUTerms_g\ SSk\\
& SAKnow_g\ SAProc_g \langle Parent\rangle
\end{array}\right\}$$

where:

- $SOp_y = \langle Op_y\rangle_{OP}$ *is the string representation of the* MAP^a *operation assigned to role* r_i *according to* $SProts$ *in* $w_{E,0}$.

- *$SUTerms_y$ is the string representation of the pairs matching formal invocation parameters of the protocol definition for role r_i, with invocation values represented by strings $SValues_y$, $SPProc_y$ and $SPKnow_y$. Each matching respects types. The string representations of the formal parameters from the protocol definition for role r_i are taken from the substring $SProts$ in $w_{E,0}$.*
- *SSk is a copy of the string $SSceneK$ in $w_{E,0}$.*
- *$SAKnow_y = \langle Saxiom_{1,y}\rangle_{AK} ... \langle Saxiom_{q,y}\rangle_{AK}\langle\rangle_{AK}$ and $SAProc_y$ is given by*
  $\langle Sproced_{1,y}((SVble_{1,y} : Stype_{1,y})...(SVble_{t_1,y} : Stype_{t_1,y})) : (Srtype_{1,y})...(Srtype_{m_1,y})\rangle_{AP}$
  ...
  $\langle Sproced_{k,y}((SVble_{k,y} : Stype_{k,y})...(SVble_{t_k,y} : Stype_{t_k}, y)) : (Srtype_{k,y})...(Srtype_{m_k,y})\rangle_{AP}\langle\rangle_{AP}$,
  $1 \leq y \leq g$.
  $SAKnow_y$ and $SAProc_y$ are, respectively, the string representations of the axioms and the decision procedures that the agent knows during the dialogue. The string representation of the set of axioms and procedures of role r_i is copied from substring $SRoles$ in $w_{E,0}$. Using recursive functions $getProced(r_i)$ and $getKnowledge(r_i)$ the axioms and procedures associated with the role r_i plus those corresponding to all the upper roles in the hierarchy of roles represented by the string $SRoles$ can be obtained. The string representation of the agent's private knowledge and agent's private decision procedures are given by the substrings $SPProc_y$ and $SPKnow_y$ from the request to create a new agent instance introduced in $w_{E,0}$.

[φ_i.2] *Mapping φ_i, is in charge of delivering at derivation time $q + 1$ all the messages $\langle Sperf_1(SValues_1), [b_1, role_1], [a, r_i]\rangle_{M,role_1,b_1}, ..., \langle Sperf_k(SValues_k), [b_k, role_k], [a, r_i]\rangle_{M,role_k,b_k}$ that are in w_E and were sent to agent $A_{r_i,(a,x)}$ by potentially different agents in previous derivation step q. But before this can happen, mapping φ_i needs to check that the string representations of the received messages are valid ones according to string SDF in w_E. In case this condition is satisfied, mapping φ_i selects the following set of rules:*

$$\left\{ \begin{array}{ll} \langle\rangle_{IM} \rightarrow & \langle Sperf_1(SValues_1)[b_1, role_1], [a, r_i]\rangle_{IM}, ..., \\ & \langle Sperf_k(SValues_k)[b_k, role_k], [a, r_i]\rangle_{IM}\langle\rangle_{IM} \end{array} \right\}$$

[φ_i.3] *If the agent $A_{r_i,(a,x)}$ is suspended and the mapping φ_i finds the substrings $\langle\rangle_{NA,r_i,a}\langle\rangle_{M,r_i,a}\langle\rangle_{AA,r_i,a}\langle\rangle_{SK,r_i,a}$ in w_E, then the agent is reactivated by applying the following rule:*

$$\left\{ \langle Suspended\rangle \rightarrow \langle Active\rangle. \right\}$$

An agent $A_{r_i,(a,x)}$ gets suspended when he starts to play a different role r_j.

4. *Mappings ψ_i, $1 \leq i \leq n$, are computable functions defined by a set of rules that use the following auxiliary functions:*

$$eval : Procedure \times 2^{Term} \times Term \times 2^{Term \times Term} \times 2^{Axiom} \rightarrow 2^{Term \times Term} \times 2^{Axiom}$$
$$eval(p, \{v_1, ..., v_n\}, \phi, MatchingP^{(u)}, Know^{(w)}) = (newMatchingP^{(e)}, newKnow^{(y)})$$

$$\begin{array}{l}
Act :: \ Operation \rightarrow Operation \\
Act(op_1 \ then \ op_2) = Act(op_1) \\
Act(op_1 \ or \ op_2) = Act(op_1) \\
Act(op_1 \ or \ op_2) = Act(op_2) \\
Act((op_1)) = (Act(op_1)) \\
Act(\alpha) = \alpha, \ if \ \alpha \in Action.
\end{array}$$

$$\begin{array}{l}
NextOp :: Operation \rightarrow Operation \cup \{NoOperation\} \\
NextOp(null) = NoOperation \\
NextOp(null \ then \ op_2) = NextOp(op_2) \\
NextOp(op_1 \ then \ null) = NextOp(op_1) \\
NextOp(op_1 \ then \ op_2) = NextOp(op_1) \ then \ op_2, \ if \ op_1 \neq null \\
NextOp(op_1 \ or \ op_2) = NextOp(op_1), \ if \ Act(op_1 \ or \ op_2) = Act(op_1) \\
NextOp(op_1 \ or \ op_2) = NextOp(op_2), \ if \ Act(op_1 \ or \ op_2) = Act(op_2) \\
NextOp((op_1)) = (NextOp(op_1)) \\
NextOp(\phi) = NextOp(subst(\phi, MatchingP^{(e)})) \\
NextOp(\alpha) = null, \ if \ \alpha \in Action
\end{array}$$

In function eval *the first component of the returned tuple is the set* $newMatchingP^{(e)}$ *resulting from adding the pair* $(\phi, p(v_1, ..., v_n))$ *to the set of matching pairs* $MatchingP^{(u)}$. *This new pair represents the association to the variable* ϕ *of the value obtained from applying procedure* p *to the values* $v_1, ..., v_n$. *The second component of the tuple is the set* $newKnow^{(y)}$ *resulting of modifying the set of axioms* $Know^{(w)}$ *after the evaluation of* $p(v_1, ..., v_n)$. *The computability of function* eval *depends on the computability of procedure* p. *By hypothesis in scene* s *all the procedures are computable functions, then function* eval *is computable.*

Taking as parameter an operation op*, function* Act *returns the first action in* op *to be performed. While function* $NextOp$ *takes as argument an operation* op *and it returns the operation that remains to be performed after the execution of the first action in* op*. According to the definition of functions* Act *and* $NextOp$*,* MAP^a *operations are interpreted in the following way:*

- null *action can be eliminated from an operation,*
- NoOperation *is returned when the current operation is* null*,*
- *operator* then *is used as a sequential composition of operations,*
- *either of the operations composed by an* or *can be chosen for its execution,*
- *the effect of the execution of an operational variable is the replacement, using function* subst*, of the operational variable for the corresponding matching value from set* $MatchingP^{(e)}$*, corresponding to the agent knowledge.*

The rules that define mappings ψ_i, $1 \leq i \leq n$, are the following:

[ψ_i.1] *If substrings $SOp = \langle op \rangle_{OC}$ and $\langle SVble_1, b \rangle_{UT}$ occur in $w_{r_i,(a,x)}$ and $Act(op) = agent(SVble_1, Sr_j, SVble_2, SVble_3, SVble_{4,1}, ..., SVble_{4,e})$ with $b \neq a$, then it means that agent $A_{r_i,(a,x)}$ wants to create a new agent instance with a different identifier. Therefore mapping ψ_i checks that the agent knows the values of the variables contained in $Act(op)$, i.e. the following substrings belong to $w_{r_i,(a,x)}$: $\langle SVble_2, SPProcs \rangle_{UT}$, $\langle SVble_3, SPKnow \rangle_{UT}$ and $\langle SVble_{4,y}, SValues_{4,y} \rangle_{UT}$, for all $1 \leq y \leq e$.*

If the previous conditions are satisfied, then ψ_i introduces in w_E at time k a string corresponding to the request to create a new agent instance. This string is added to w_E rewriting the symbol $\langle \rangle_{NA,r_i,a}$. Besides, mapping ψ_i introduces the pairs matching formal and real parameters used for the creation of the new agent instance, and it actualizes the operation being enacted by the agent $A_{r_i,(a,x)}$.

So mapping ψ_i selects the following set of rules:

$$\left\{ \begin{array}{l} \langle \rangle_{NA,r_i,a} \rightarrow \quad \langle agent(b, r_j, [r_i, a, x], SPProcs, SPKnow, SValues \rangle_{NA,r_i,a} \\ \qquad\qquad\qquad \langle \rangle_{NA,r_j,b} \langle \rangle_{M,r_j,b} \langle \rangle_{AA,r_j,b}, \\ \langle \rangle_{UT} \rightarrow \langle SVble_1, b \rangle_{UT} \langle SVble_2, SPProcs \rangle_{UT} \langle SVble_3, SPKnow \rangle_{UT} \langle SVble_4, SValues \rangle_{UT} \langle \rangle_{UT}, \\ \langle op \rangle_{OP} \rightarrow \langle newop \rangle_{OP} \mid newop = NextOp(op) \end{array} \right\}$$

where symbols $\langle \rangle_{NA,r_j,b} \langle \rangle_{M,r_j,b} \langle \rangle_{AA,r_j,b}$ will be used by the new agent $A_{r_j,(b,k+1)}$ for introducing in w_E strings which correspond to: requests to create new agent instances, requests to active agents, and messages sent.

[ψ_i.2] *If the substrings $\langle Active \rangle$, $\langle op \rangle_{OC}$ and $\langle SVble_1, a \rangle_{UT}$ occur in $w_{r_i,(a,x)}$ and $Act(op) = agent(SVble_1, Sr_j, SVble_2, SVble_3, SVble_{4,1}, ..., SVble_{4,e})$, then the agent $A_{r_i,(a,x)}$ wants to create a new agent instance with his same identifier. For example it can be the case that the agent wants to change of role or to play the same role again (recursive call). Therefore mapping ψ_i verifies that the following substrings belong to $w_{r_i,(a,x)}$: agent's private decision procedures $SMyProc$, agent's axioms $SMyKnow$, and agent's matching pairs $\langle SVbles_2, SPProcs \rangle_{UT}$, $\langle SVbles_3, SPKnow \rangle_{UT}$, and $\langle SVbles_{4,y}, SValues_{4,y} \rangle_{UT}$ for all $1 \leq y \leq e$.*

If the preconditions are satisfied then ψ_i rewrites symbol $\langle \rangle_{NA,r_i,a}$ in w_E at time q introducing a string corresponding to the request to create a new agent instance followed by symbols $\langle \rangle_{NA,r_j,a} \langle \rangle_{M,r_j,a} \langle \rangle_{AA,r_j,a}$. Also mapping ψ_i deletes symbols $\langle \rangle_{M,r_i,a}$, $\langle \rangle_{AA,r_i,a}$ from w_E and changes agent's current state from active to suspended. These rewritings are necessary because after changing to a new role agent $A_{r_i,(a,x)}$ has to interrupt the execution of the operation associated with its current role r_i. Only after the agent finishes performing the role r_j it can reassume the execution of role r_i. Besides mapping ψ_i actualizes the agent's current operation and introduces the pairs matching formal and real parameters used for the creation of the new agent instance.

So mapping ψ_i selects the following set of rules:

$$\left\{\begin{array}{l} \langle Active\rangle \to \langle Suspended\rangle, \\ \langle\rangle_{M,r_i,a} \to \lambda, \\ \langle\rangle_{AA,r_i,a} \to \lambda, \\ \langle\rangle_{UT} \to \langle SVble_1, b\rangle_{UT}\langle SVble_2, SPProcs\rangle_{UT}\langle SVble_3, SPKnow\rangle_{UT} \\ \qquad\qquad \langle SVble_{4,1}, SValues_{4,1}\rangle_{UT}...SVble_{4,e}, SValues_{4,e}\rangle_{UT}\langle\rangle_{UT} \\ \langle\rangle_{NA,r_i,a} \to \langle agent(b, r_j, [r_i, a, x], SPProcs\ SMyProcs, \\ \qquad\qquad SPKnow\ SMyKnow, SValues))_{NA,r_i,a}\langle\rangle_{NA,r_j,a}\langle\rangle_{M,r_j,a}\langle\rangle_{AA,r_j,a} \\ \langle op\rangle_{OP} \to \langle newop\rangle_{OP} \mid newop = NextOp(op) \end{array}\right\}$$

where symbols $\langle\rangle_{NA,r_j,a}\langle\rangle_{M,r_j,a}\langle\rangle_{AA,r_j,a}$ *could be rewritten by the new agent* $A_{r_j,(a,q)}$ *in case it wants to introduce in* w_E *requests to create new agent instances, requests to reactivate agents, and send messages.*

[ψ_i.3] *If* $w_{r_i,(a,x)}$ *contains the substrings* $\langle op\rangle_{OP}$, $\downarrow \langle Sperf(Svalue_1, ..., Svalue_h)[b, r_j], [c, r_l]\rangle_{IM}$ *and* $\langle SVble_2, b\rangle_{UT}$ *and* $Act(op) = (perf(Vble_1, .., Vble_h) \Leftarrow agent(Vble_2, r_j))$ *and* $((c = \text{"\_"}) \wedge (r_l = \text{"\_"})) \vee ((c = \text{"\_"}) \wedge (r_l = r_i)) \vee ((c = a) \wedge (r_l = r_i))$ *then the reception of a message is simulated by deleting in* $w_{r_i,(a,x)}$ *the string representation of the message. Mapping* ψ_i *selects the following rules:*

$$\left\{\begin{array}{l} \downarrow \langle Sperf(SValue_1, ..., SValue_h), [b, r_j], [c, r_l]\rangle_{IM,r_i,a} \to\downarrow, \\ \langle op\rangle_{OP} \to \langle newop\rangle_{OP} \mid newop = nextOp(op) \end{array}\right\}$$

While if the previous conditions are satisfied but substring $\langle SVble_2, b\rangle_{UT}$ *is not in* $w_{r_i,(a,x)}$ *it means that the received message was sent by an unknown agent with identifier b. Then mapping* ψ_i *selects the following rules:*

$$\left\{\begin{array}{l} \downarrow \langle Sperf(SValue_1, ..., SValue_h), [b, r_j], [c, r_l]\rangle_{IM,r_i,a} \to\downarrow, \\ \langle\rangle_{UT} \to \langle SVble_2, b\rangle_{UT}\langle\rangle_{UT}, \\ \langle op\rangle_{OP} \to \langle newop\rangle_{OP} \mid newop = nextOp(op) \end{array}\right\}$$

where $SVble_2$ *and b are the string representations of a variable and value of the same type.*

[ψ_i.4] *If* $w_{r_i,(a,x)}$ *contains the substrings* $\langle op\rangle_{OP}$, $\langle SVble_2, b\rangle_{UT}$ *and* $\langle SVble_{1,1}, SValue_{1,1}\rangle_{UT}...$ $\langle SVble_{1,e}, SValue_{1,e}\rangle_{UT}$ *such that* $Act(op) = (perf(Vble_1, .., Vble_e) \Rightarrow agent(Vble_2, r_j))$, *then the agent is sending a message. So mapping* ψ_i *copies the message in* w_E *and actualizes the agent's current operation choosing the following set of rules:*

$$\left\{\begin{array}{l} \langle\rangle_{M,r_i,a} \to \langle Sperf(SValue_1, ..., Svalue_e)[a, r_i][b, r_j]\rangle_{M,r_i,a}\langle\rangle_{M,r_i,a}, \\ \langle op\rangle_{OP} \to \langle newop\rangle_{OP} \mid newop = NextOp(op) \end{array}\right\}$$

In case the message is addressed to one agent then $r_j \neq \_$ *and* $b \neq \_$. *But if the message is sent to all the agents with role* r_j *then* $b = \_$ *and it is called broadcast communication. It can also happen that the message is sent to all the agents participating in the conversation, in which case it is called multicast communication and* $r_j = \_$ *and* $b = \_$.

[ψ_i.5] *If the following substrings are present in $w_{r_i,(a,x)}$: $\langle op\rangle_{OP}$ and $\langle (Srtype_1) :: Sproced(Svble_1, Stype_1)...(Svble_h, Stype_h)\rangle_{AP}$, $\langle Svble_j, Svalue_j\rangle_{UT}$, for every $1 \leq j \leq h$, and $Act(op) = (SVble_2 = SProced(SVble_3))$, then mapping ψ_i invocates procedure proced obtaining: a new value represented by the string SNewValue, and a set of axioms given by string NewAgentK corresponding to the new agent's private knowledge base.*

If there is a substring $\langle SVble_1, SOldValue_1\rangle_{UT}$ in $w_{r_i,(a,x)}$, it means that the variable $SVble_1$ was given the value $SOldValue_1$ previous to this assignation, then ψ_i modifies the old value by the new one selecting the following rule:

$$\left\{ \langle SVble_1, SOldValue_1\rangle_{UT} \to \langle SVble_1, SNewValue\rangle_{UT}, \right\}$$

If there is no substring $\langle SVble_1, SOldValue_1\rangle_{UT}$ in $w_{r_i,(a,x)}$, it means that no value was assigned to the variable $SVble_1$ previous to this assignation, then ψ_i selects the set of rules:

$$\left\{ \langle\rangle_{UT} \to \langle SVble_1, SNewValue\rangle_{UT}\langle\rangle_{UT} \right\}$$

If the effects of the execution of the decision procedure over the agent's private knowledge are the deletion of axioms $\{f_1, ..., f_n\}$, the modification of axioms $\{g_1, ..., g_d\}$ for new axioms $\{h_1, ..., h_d\}$, and the addition of new axioms $\{i_1, ..., i_p\}$, then ψ_i chooses the following rules:

$$\left\{ \begin{array}{l} \langle f_1\rangle_{AK} \to \lambda, \ ..., \ \langle f_n\rangle_{AK} \to \lambda, \\ \langle g_1\rangle_{AK} \to \langle h_1\rangle_{AK}, \ ..., \ \langle g_d\rangle_{AK} \to \langle h_d\rangle_{AK}, \\ \langle\rangle_{AK} \to \langle i_1\rangle_{AK}, ..., \langle i_p\rangle_{AK}\langle\rangle_{AK}. \end{array} \right\}$$

In addition the agent's operation is actualized applying the rule:

$$\left\{ \langle op\rangle_{OP} \to \langle newop\rangle_{OP} \mid newop = NextOp(op) \right\}$$

[ψ_i.6] *An agent $A_{r_i,(a,x)}$ is inactive when substrings $\langle Active\rangle$ and $\langle null\rangle_{OP}$ occur in $w_{r_i,(a,x)}$. The mapping ψ_i checks the substring $[b, r_j, k]$ in $w_{r_i,(a,x)}$ which provides information about the agent $A_{r_j,(b,k)}$ that created the agent instance $A_{r_i,(a,x)}$ with $k \leq x$:*

(a) *In case the identifier b is equal to the identifier a it means that the creator of agent $A_{r_i,(a,x)}$ remains suspended, waiting to reassume its activity after agent $A_{r_i,(a,x)}$ becomes inactive. Then mapping ψ_i introduces in w_E the symbols that could allow agent $A_{r_j,(a,k)}$ to introduce: requests to create new agent instances, requests to reactivate suspended agents, and messages. Meanwhile mapping ψ_i eliminates from w_E the symbols that could allowed agent $A_{r_i,(a,x)}$ to perform the previously mentioned actions. So the following rules are selected by mapping ψ_i:*

$$\left\{ \begin{array}{l} \langle\rangle_{AA,r_i,a} \to \langle\rangle_{NA,r_j,a}\langle\rangle_{M,r_j,a}\langle\rangle_{AA,r_j,a}, \\ \langle\rangle_{NA,r_i,a} \to \lambda, \\ \langle\rangle_{AA,r_i,a} \to \lambda, \\ \langle\rangle_{M,r_i,a} \to \lambda \end{array} \right\}$$

(b) *In case the identifier b is different from the identifier a, no reactivation of the creator needs to be done. Then mapping ψ_i eliminates from w_E the symbols that allowed agent*

$A_{r_i,(a,x)}$ to introduce: requests to create new agent instances, requests to reactivate suspended agents, and messages. Then the following rules are selected by mapping ψ_i:
$$\left\{ \begin{array}{l} \langle\rangle_{NA,r_i,a} \to \lambda, \\ \langle\rangle_{AA,r_i,a} \to \lambda, \\ \langle\rangle_{M,r_i,a} \to \lambda \end{array} \right\}$$

5. *Evolution rules P_E are defined in the following way:*

   [P_E.1] *At time q evolution rules P_E deletes the substrings from w_E that correspond to: requests to create new agent instances, requests to reactivate agents, and messages interchanged at time $q-1$.*

When we are interested only in strings over some alphabet T, hence in the language $L_E(\Sigma, \sigma_0) \cap T^*$, then we speak about an Extended Conversational *Extended Reproductive Eco-Grammar (EConvEREG)* system. We denote by $EConvEREG_n$, $n \geq 1$ the family of languages $L_E(\Sigma, \sigma_0) \cap T^*$ where $\Sigma \in ConvEREG_n$.

4.3.3 An example of dialogue protocol

CD Grammar systems with memories, introduced in [CVJJP94], are finite sets of semi-conditional grammars, each of them associated with a stack called memory. During the common derivation, the component grammars send messages to each other and use their memories for storing/erasing messages (words) they receive. The current sentential form is controlled both by the contents of the memories and the context conditions associated with the productions.

In [AJR01] the use of CD Grammar systems with memories was exemplified with the simulation of a dialogue protocol: a 2-party dialogue in a film shop, with one of the speaker (customer) trying to buy a film and the other speaker (clerk) trying to guess the film that the customer needs in order to sell it. For modeling this dialogue protocol they propose a CD Grammar system with memories $\Gamma = (T, G_S, G_C, S)$ where G_C and G_S are the grammars for customer C and clerk S, respectively.

The configuration of Γ is given by $(w, \upsilon_S, \upsilon_C)$ where:

- w is used for the interchange of information or messages and for storing information shared by S and C. It can be seen as the dialogic framework.
- υ_S and υ_C record the mental state of S and C, respectively. It can be knowledge, satisfied and unsatisfied goals, or other information relevant for participating in the dialogue.

In figure 4.2 we introduce a dialogue instance that is specified by the dialogue protocol introduced in [AJR01].

We exemplify the use of *ConvEREG* systems for the specification of dialogue protocols with the protocol from [AJR01]. We base the scene $Sell$ on the information-seeking dialogue from the typology introduced by Walton and Krabbe in [WK95]. Their categorization identifies six primary types of dialogues: information-seeking, inquiry, persuasion, negotiation, deliberation and eristic dialogues. Information-Seeking Dialogues are those where one participant seeks the answer to some question(s) from another participant, who is believed by the first to know the answer(s). We

Speaker	Utterance
C:	I need a film.
S:	Which kind of film?
C:	One for 36 pictures.
S:	Which brand and speed?
C:	Speed? What do you mean?
S:	Where will you take your pictures?
C:	Outside.
S:	I suggest the Kodak 100.
C:	OK.

Figure 4.2: A dialogue instance between a customer C and a clerk S

define the scene $Sell$ as an information-seeking dialogue where the role of the information seeker is taken by the clerk and the role of the information provider is taken by the customer. Therefore, the scene $Sell$ is defined as:

$$Sell = \langle InitAgs, Roles, Protocols, SK, DF, MergeK_{EREG}\rangle$$

with components specified in the following way:

- $Roles = \langle Customer, CustomerP, CustomerK, ur_c, t\rangle\langle Clerk, ClerkP, ClerK, ur_e\rangle$ $\langle IP, IPProc, IPKnow, ur_{IP}\rangle\langle IS, ISProc, ISKnow, ur_{IS}\rangle$ correspond to Customer, Clerk, Information Provider and Information Seeker roles.

  1. For the customer role the knowledge base is $CustomerK = \emptyset$ and the set of decision procedures is the minimal possible set of actions required to join the scene: $CustomerP = \{Identifier :: choose\_clerk(t, string)\}$. No upper roles are selected, $ur_c = \emptyset$. Extra parameter t is considered to represent the topic under discussion.
  2. For the clerk role the knowledge base is given by the set $ClerkK = Stock \cup Selling\_strategies \cup Competence\_strategies \cup Competence\_prices$ and the set of known procedures is given by the set $ClerkP = \{Real :: price(film, String)$, $List :: characteristics(film, String)\}$. No upper roles are selected, $ur_e = \emptyset$.
  3. While the roles customer and clerk are specific to the dialogue instance we model the roles IS and IP as generic definitions. They can be used to simulate any dialogue between two agents working together to find the answer to a question that neither of them know. For example, customer and clerk roles could be replaced for Patient and Doctor roles, the topic of the conversation could change from finding an adequate film for a set of buyer requirements, to diagnosing the disease that corresponds to a set of patient symptoms. Therefore, for the IP role no knowledge is assigned , $IPKnow = \emptyset$, and the minimal set of decision procedures required to performed the assigned protocol is given by $IPProc = \left\{ \begin{array}{l} (String, String) :: get\_answer(History, PairSet), \\ Boolean :: check\_adequacy(r, String), null :: pay(amount, Real) \end{array} \right\}$, where $History \in PairSet$ is a set used to register information interchanged during the

dialogue and type $PairSet = \{(\alpha,\beta) \mid \alpha,\beta \in String\}$. No upper roles are selected, $ur_{IP} = \emptyset$.

4. With the same criteria we choose no axioms for role IS, $ISKnow = \emptyset$, the minimum set of decision procedures $ISProc = \left\{ \begin{array}{l} String :: next\_question(History, PairSet), \\ String :: final\_answer(t, String), (History, PairSet) \end{array} \right\}$ and no upper roles, $ur_{IS} = \emptyset$.

- *Protocols* for Customer, Clerk, IP and IS roles are presented in figure 4.3. In scene *Sell* one speaker, the seller, seeks the characteristics of the item that a second speaker, the buyer, wants to buy. The scene comprises four role definitions: *Customer, Clerk, InformationSeeker (IS) and InformationProvider (IP)*. The customer initiates the dialogue choosing a clerk, changing its role to provider. The clerk role waits for a customer to ask for an item t to change to the seeker role.

  The core of the dialogue is between a provider who wants to buy an item t and a seeker who inquires item characteristics to determine the product to sell. The results of the interchanges of questions q from the seeker and answers a from the provider are collected during the dialogue by both roles in their respective $History$ sets.

  The IP role is defined in four cases separated by the choice operator. In the first case, the information provider receives from an information seeker a request for answering a question q. The information provider will attempt to answer the request and will save the pair $(q =?)$ in the $History$ set. The appropriate answer is determined in cases 2, 3 and 4. Second case is triggered when the information provider has a direct answer a for the request q, in which case this answer is returned to the seeker, incorporating $(q = a)$ in $History$. In the third case more information is required, so the provider becomes a seeker for this new information. Upon obtaining this extra information, the speaker reverts back from a seeker to a provider again. This is a clear example of a combination of dialogue types, in this case the embedding of Information-Seeking dialogues is possible through the changing of role. In the final case the provider receives, from the seeker, the answer r that matches the characteristics agreed in $History$ over the item t. He adds the pair $(solution = r)$ to $History$ and then he decides if he accepts or rejects r.

  The information seeker (IS) protocol is essentially the complement of the information provider protocol, and there are four corresponding cases. In the first case, the seeker sends a request $newq$ for information to the provider adding $(newq =?)$ to $History$. The second case deals with the response from the provider, which actualizes the set $History$ and restarts the protocol. In the third case, the speaker decides to be more explicit and clearer about what he is asking for to the other speaker. After providing the required information he reassume the role of seeker. Lastly, he can decide that the collected information contained in $History$ is enough to determine the item to sell. In this case, he informs the provider with the variable r which is the object that matches the agreed characteristics and he incorporates $(solution = r)$ to $History$. He then waits for the decision taken by the IP, which can be acceptance or rejection of r.

- $SK = \{currency = euro\}$ indicates that all the speakers know that the accepted currency is the euro.

- $DF = \left\{ \begin{array}{l} ask(t, String), request(q, String), inform(q, String)(a, String), \\ solution(r, String), accept(), refuse() \end{array} \right\}$ is the set of common locutions.

- The scene is initialized with one speaker called Joe as the customer, and one speaker called Ana as the clerk chosen by Joe. Therefore, the set of initial speakers associated with the scene is defined as $InitAgs = \{agent(Joe, Customer, JoeProc, JoeK, film)\ agent(Ana, Clerk, \emptyset, \emptyset)\}$. The set $JoeProc = \{Real :: estimate\_price(film, String)\}$ is provided to the speaker Joe as his private decision procedures. This means that Joe knows for some photograph films the estimated price in the market. Not all the customers know this, so it is considered part of the private decision procedures of Joe. The private knowledge of Joe corresponds to the characteristics of the film he wants to buy, the money he has, and an empty set of dialogue history registers, so that $JoeKnow = \left\{ \begin{array}{l} object = film, quantity = 1, photos = 36, money = 5, \\ purpose = summer\_holidays, History = \emptyset \end{array} \right\}$. While Ana is considered a new employee in the shop without much experience in the field. Hence, she only knows what clerks are supposed to know when they start working in the shop, the knowledge represented by strings *ClerkProc* and *ClerkK*. Ana has no private decision procedures or knowledge.

From the MAP^a definition $Sell$, we get $\Sigma_{sell} \in ConvEREG_4$ defined in the following way:

$$\Sigma_{sell} = (E, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4)$$

where population $\mathcal{A}_1$ corresponds to the seller role, $\mathcal{A}_2$ the clerk role, $\mathcal{A}_3$ the IP role and $\mathcal{A}_4$ the IS role, and $W_{i,0} = \{\langle Parent \rangle\}$, for every $1 \leq i \leq 4$.

The initial configuration of the Σ_{sell} system is given by:

$$\sigma_{\Sigma_{sell},0} = (w_{E,0}, \{\langle Parent \rangle\}, \{\langle Parent \rangle\}, \{\langle Parent \rangle\}, \{\langle Parent \rangle\})$$

where:

- $w_{E,0} = Roles + +Protocols + +ShareK + +DFramework + +\alpha$, where:

  - $\begin{array}{ll} Roles = & \langle Customer, CustomerP, CustomerK, t\rangle_R \langle Clerk, ClerkP, ClerkK\rangle_R \\ & \langle IP, IPProc\rangle_R \langle IS, ISProc\rangle_R \end{array}$
    where:
    * $CustomerP = \langle Identifier :: choose\_clerk(t, String)\rangle_{RP}$,
    * $CustomerK = \langle History = \emptyset\rangle_{RK}$,
    * $ClerkP = \langle Real :: price(film, String)\rangle_{RP}\ \langle List :: characteristics(film, String)\rangle_{RP}$,
    * $ClerkK = \langle stock\rangle_{RK}\ \langle selling\_strategies\rangle_{RK}\ \langle competence\_strategies\rangle_{RK}\ \langle competence\_prices\rangle_{RK}$,

$$\begin{array}{l} agent(id_1,\ Customer, Proc^{(a)}, K^{(b)}, t) :: \\ id_2 = choose\_clerk(t)\ then \\ ask(t) \Longrightarrow agent(id_2, IS)\ then \\ agent(id_1,\ IP,\ Proc^{(a)}, K^{(b)}). \end{array}$$

$$\begin{array}{l} agent(id_1, Clerk, Proc^{(c)}, K^{(d)}) :: \\ ask(t) \Leftarrow agent(id_1, Customer) \\ then\ agent(id_1, IS, Proc^{(c)}, K^{(d)}). \end{array}$$

$$\begin{array}{l} agent(id_1,\ IP,\ Proc^{(t)}, K^{(z)}) :: \\ \left[\begin{array}{l} request(q) \Longleftarrow agent(id_2,\ IS)\ then \\ History = History \cup \{(q = ?)\}\ then \\ agent(id_1,\ IP,\ Proc^{(t)}, K^{(z)}) \end{array}\right] \\ or \\ \left[\begin{array}{l} (q, a) = get\_answer(History \cup K^{(z)})\ then \\ History = History \cup \{(q = a)\}\ then \\ inform(q,\ a) \Longrightarrow agent(id_2,\ IS)\ then \\ agent(id_1, IP, Proc^{(t)}, K^{(z)}) \end{array}\right] \\ or \\ \left[\begin{array}{l} agent(id_1,\ IS, Proc^{(m)}, K^{(s)})\ then \\ agent(id_1,\ IP,\ Proc^{(t)}, K^{(z)}) \end{array}\right] \\ or \\ \left[\begin{array}{l} solution(r) \Longleftarrow agent(id_2\ IS)\ then \\ History = History \cup \{(solution = r)\}\ then \\ a = checkadequacy(r)\ then \\ \left[\begin{array}{l} accept() \Longrightarrow agent(id_2, IS)\ or \\ refuse() \Longrightarrow agent(id_2, IS) \end{array}\right] \end{array}\right] \end{array}$$

$$\begin{array}{l} agent(id_1,\ IS,\ Proc^{(m)}, K^{(s)}) :: \\ \left[\begin{array}{l} newq = next\_question(History \cup K)\ then \\ History = History \cup \{(newq = ?)\}\ then \\ request(newq) \Longrightarrow agent(id_2,\ IP)\ then \\ agent(id_1\ \ IS, Proc^{(m)}, K^{(s)}) \end{array}\right] \\ or \\ \left[\begin{array}{l} inform(q,\ a) \Longleftarrow agent(id_2,\ IP)\ then \\ History = History \cup \{(q = a)\}\ then \\ agent(id_1,\ IS,\ Proc^{(m)}, K^{(s)}) \end{array}\right] \\ or \\ \left[\begin{array}{l} agent(id_1,\ IP, Proc^{(t)}, K^{(z)})\ then \\ agent(id_1,\ IS,\ Proc^{(m)}, K^{(s)}) \end{array}\right] \\ or \\ \left[\begin{array}{l} r = final\_answer(t, History)\ then \\ solution(r) \Longrightarrow agent(id_2, IP)\ then \\ History = History \cup \{(solution = r)\}\ then \\ \left[\begin{array}{l} accept() \Longleftarrow agent(id_1, IP)\ or \\ refuse() \Longleftarrow agent(id_1, IP) \end{array}\right] \end{array}\right] \end{array}$$

Figure 4.3: A MAP^a scene for a dialogue in a shop

  * $IPProc = \langle (String, String) :: get\_answer(History, PairSet)\rangle_{RP}$
    $\langle Boolean :: check\_adequacy(r, String)\rangle_{RP}$
    $\langle null :: pay(amount, Real)\rangle_{RP}$,
  * $ISProc = \langle String :: next\_question(History, PairSet)\rangle_{RP}$
    $\langle String :: final\_answer(t, String)(History, PairSet)\rangle_{RP}$.

- $Protocols = \langle Customer, 1, (t : string), Op_{Customer}\rangle_{PC}\ \langle Clerk, 2, Op_{Clerk}\rangle_{PC}\ \langle IP, 3, Op_{IP}\rangle_{PC}\ \langle IS, 4, Op_{IS}\rangle_{PC}$ where $Op_{Customer}$, Op_{Clerk}, Op_{IP}, and Op_{IS} are the MAP^a operations introduced in figure 4.3 for roles Customer, Clerk, IP and IS, respectively.
- $ShareK = \langle currency = euro\rangle_{SK}$ fixes the currency that is accepted in the shop.
- $DFramework = \langle ask(x, String)\rangle_{DF}\ \langle request(x, String)\rangle_{DF}\ \langle accept\rangle_{DF}\ \langle refuse\rangle_{DF}\ \langle inform(x, String)(y, String)\rangle_{DF}\ \langle solution(x, String)\rangle_{DF}$ fixes the dialogic framework as the minimal set of locutions that speakers involved in the dialogue need to know.
- $\alpha = \langle agent(Joe, 1, [\_, \_, \_], JoeP, JoeK, film)\rangle_{NA,\_,\_}\langle\rangle_{NA,1,Joe}\langle\rangle_{M,1,Joe}\langle\rangle_{AA,1,Joe}$ $\langle agent(Ana, 2, [\_, \_, \_])\rangle_{NA,\_,\_}\langle\rangle_{NA,2,Ana}\langle\rangle_{M,2,Ana}\langle\rangle_{AA,2,Ana}$ where:

$JoeP = \langle Real :: estimate\_price(film, String)\rangle_{AK}\langle\rangle_{AK}$ and $JoeK = \langle object = film\rangle_{AK}$ $\langle quantity = 1\rangle_{AK}$ $\langle photos = 36\rangle_{AK}$ $\langle purpose = summer\_holidays\rangle_{AK}$ $\langle money = 5\rangle_{AK}\langle History = \emptyset\rangle_{AK}\langle\rangle_{AK}$. *JoeP* is the string representation of *JoeProc*, and *JoeK* is the string representation of *JoeKnow*.

1. In derivation step 1, two new agent instances are created according to the requests introduced in the initial configuration $w_{E,0}$; one agent instance corresponding to population $\mathcal{A}_1$, and the other corresponding to population $\mathcal{A}_2$. Then by rules **[$\varphi_{1,(Joe,1)}$.1]**, **[$\varphi_{2,(Ana,1)}$.1]** and **[P_E.1]** the following configuration is obtained:

$$\sigma_{\Sigma_{sell},1} = (w_{E,1}, \{w_{1,(Joe,1)}\}, \{w_{2,(Ana,1)}\}, \{\langle Parent\rangle\}, \{\langle Parent\rangle\}) \text{ where:}$$

- $w_{E,1} = \varepsilon\ \langle\rangle_{NA,1,Joe}\langle\rangle_{M,1,Joe}\langle\rangle_{AA,1,Joe}\langle\rangle_{NA,2,Ana}\langle\rangle_{M,2,Ana}\langle\rangle_{AA,2,Ana}$, where $\varepsilon = Roles + +Protocols + +ShareK + +DFramework$.
- $w_{1,(Joe,1)} = Joe[\_,\_,\_] \downarrow \langle\rangle_{IM}\langle id_2 = choose\_clerk(t)\ then\ \gamma_{1,1,1}.\rangle_{OP}\langle Active\rangle\langle id_1, Joe\rangle_{UT}$ $\langle Proc^{(a)}, JoeP\rangle_{UT}\langle K^{(b)}, JoeK\rangle_{UT}\langle t, film\rangle_{UT}\langle\rangle_{UT}\ SK\ AK_1\ \langle\rangle_{AK}\ AProc_1\ \langle Parent\rangle$ where:
  - SK and AK_1 are, respectively, copies of strings *ShareK* and *JoeK* in $w_{E,0}$.
  - $AProc_1 = \langle Identifier :: choose\_clerk(t, String)\rangle_{AP}$ $\langle Real :: estimate\_price(film, String)\rangle_{AP}$ is the string representation of the decision procedures for role Customer, *CustomerP*, and the private decision procedures of Joe, *JoeProc*.
- $w_{2,(Ana,1)} = Ana[\_,\_,\_] \downarrow \langle\rangle_{IM}\langle ask(t) \Leftarrow agent(id_1, Customer)\ then\ \gamma_{1,2,1}.\rangle_{OP}\langle Active\rangle$ $\langle id_1, Ana\rangle_{UT}\langle\rangle_{UT}\ SK\ AK_2\ \langle\rangle_{AK}\ AProc_2\ \langle Parent\rangle$ where:
  - SK, AK_2 and $AProc_2$ are, respectively, copies of the strings *ShareK, ClerkK* and *ClerkP* in $w_{E,0}$.

2. In derivation step 2, the execution of action *Ana=choose_clerk(film)* by agent $A_{1,(Joe,1)}$ is simulated by the execution of rule **[$\psi_{1,(Joe,1)}$.5]**, getting the configuration:

$$\sigma_{\Sigma_{sell},2} = (w_{E,2}, \{w_{1,(Joe,1)}\}, \{w_{2,(Ana,1)}\}, \{\langle Parent\rangle\}, \{\langle Parent\rangle\}) \text{ where:}$$

- $w_{E,2} = w_{E,1}$
- $w_{1,(Joe,1)} = Joe[\_,\_,\_] \downarrow \langle\rangle_{IM}\langle ask(t) \Rightarrow agent(id_2, Clerk)\ then\ \gamma_{1,1,2}.\rangle_{OP}\langle Active\rangle$ $\langle id_1, Joe\rangle_{UT}\langle Proc^{(a)}, JoeP\rangle_{UT}\langle K^{(b)}, JoeK\rangle_{UT}\langle t, film\rangle_{UT}\langle id_2, Ana\rangle_{UT}\langle\rangle_{UT}\ SK$ $AK_1\ \langle clerk = Ana\rangle_{AK}\ \langle\rangle_{AK}\ AProc_1\ \langle Parent\rangle$
- $w_{2,(Ana,1)}$ does not change.

3. In derivation step 3, the execution of action *ask(film)⟹ agent(Ana, IS)* by agent $A_{1,(Joe,1)}$ is simulated by rule **[$\psi_{1,(Joe,1)}$.4]** getting the configuration:

$$\sigma_{\Sigma_{sell},3} = (w_{E,3}, \{w_{1,(Joe,1)}\}, \{w_{2,(Ana,1)}\}, \{\langle Parent\rangle\}, \{\langle Parent\rangle\}) \text{ where:}$$

- $w_{E,3} = \varepsilon\ \langle\rangle_{NA,1,Joe}\langle ask(film), [Joe, 1], [Ana, 2]\rangle_{M,1,Joe}\langle\rangle_{M,1,Joe}\langle\rangle_{AA,1,Joe}$ $\langle\rangle_{NA,2,Ana}\langle\rangle_{M,2,Ana}\langle\rangle_{AA,2,Ana}$
- $w_{1,(Joe,1)} = Joe[\_,\_,\_] \downarrow \langle\rangle_{IM}\langle agent(id_1, IS, Proc^{(c)}, K^{(d)}).\rangle_{OP}\langle Active\rangle$ $\langle id_1, Joe\rangle_{UT}\langle Proc^{(a)}, JoeP\rangle_{UT}\langle K^{(b)}, JoeK\rangle_{UT}\langle t, film\rangle_{UT}\langle id_2, Ana\rangle_{UT}\langle\rangle_{UT}\ SK\ AK_1$ $\langle clerk = Ana\rangle_{AK}\ \langle\rangle_{AK}\ AProc_1\ \langle Parent\rangle$
- $w_{2,(Ana,1)}$ does not change.

4. In derivation step 4, the simultaneous execution of actions $agent(Joe, IP, JoeProc, JoeKnow)$ by agent $A_{1,(Joe,1)}$ and the reception by agent $A_{2,(Ana,1)}$ of message $ask(film)$ from agent $A_{1,(Joe,1)}$ are simulated. By application of rules $[\boldsymbol{\psi_{1,(Joe,1)}.2}]$, $[\boldsymbol{\varphi_{2,(Ana,1)}.2}]$, and $[\boldsymbol{P_E.1}]$ the following configuration is obtained:

$$\sigma_{\Sigma_{sell},4} = (w_{E,4}, \{w_{1,(Joe,1)}\}, \{w_{2,(Ana,1)}\}, \{\langle Parent\rangle\}, \{\langle Parent\rangle\}) \text{ where:}$$

- $w_{E,4} = \varepsilon\ \langle agent(Joe, 3, [Joe, 1, 1], JoeP, JoeK)\rangle_{NA,1,Joe}\langle\rangle_{NA,3,Joe}$ $\langle\rangle_{M,3,Joe}\langle\rangle_{AA,3,Joe}\langle\rangle_{NA,2,Ana}\langle\rangle_{M,2,Ana}\langle\rangle_{AA,2,Ana}$
- $w_{1,(Joe,1)} = Joe[\_,\_,\_] \downarrow \langle\rangle_{IM}\langle null\rangle_{OP}\langle Suspended\rangle\langle id_1, Joe\rangle_{UT}\langle Proc^{(a)}, JoeP\rangle_{UT}$ $\langle K^{(b)}, JoeK\rangle_{UT}\langle t, film\rangle_{UT}\langle id_2, Ana\rangle_{UT}\langle\rangle_{UT}\ SK\ AK_1\langle clerk = Ana\rangle_{AK}\langle\rangle_{AK}AProc_1\ \langle Parent\rangle$
- $w_{2,(Ana,1)} = Ana[\_,\_,\_] \downarrow \langle ask(film), [Joe, 1], [Ana, 2]\rangle_{IM}\langle\rangle_{IM}$ $\langle\ ask(t) \Leftarrow agent(id_1, Customer)\gamma_{1,2,4}).\rangle_{OP}\langle Active\rangle\langle id_1, Ana\rangle_{UT}\langle\rangle_{UT}\ SK\ AK_2\langle\rangle_{AK}$ $AProc_2\ \langle Parent\rangle$

5. In derivation step 5, the simultaneous creation of a new agent instance from population $\mathcal{A}_3$ and the execution of operation $ask(film) \Leftarrow agent(Joe, Customer)$ by agent $A_{2,(Ana,1)}$ are simulated by rules $[\boldsymbol{\psi_{2,(Ana,1)}.3}]$, $[\boldsymbol{\varphi_{3,(Joe,1)}.1}]$ and $[\boldsymbol{P_E.1}]$. The following configuration is obtained:

$$\sigma_{\Sigma_{sell},5} = (w_{E,5}, \{w_{1,(Joe,1)}\}, \{w_{2,(Ana,1)}\}, \{w_{3,(Joe,1)}\}, \{\langle Parent\rangle\}) \text{ where:}$$

- $w_{E,5} = \varepsilon\ \langle\rangle_{NA,3,Joe}\langle\rangle_{M,3,Joe}\langle\rangle_{AA,3,Joe}\ \langle\rangle_{NA,2,Ana}\langle\rangle_{M,2,Ana}\langle\rangle_{AA,2,Ana}$
- $w_{1,(Joe,1)}$ does not change,
- $w_{2,(Ana,1)} = Ana[\_,\_,\_]\ \downarrow \langle\rangle_{IM}\langle agent(id_2, IS, Proc^{(e)}, K^{(d)} \cup \{History = \emptyset\}).\rangle_{OP}\langle Active\rangle$ $\langle id_1, Ana\rangle_{UT}\langle t, film\rangle_{UT}\langle id_1, Joe\rangle_{UT}\langle\rangle_{UT}\ SK\ AK_2\langle\rangle_{AK}\ AProc_2\ \langle Parent\rangle$
- $w_{3,(Joe,1)} = Joe[Joe, 1, 1] \downarrow \langle\rangle_{IM}\langle Op_{IP}\rangle_{OP}\langle Active\rangle\langle id_1, Joe\rangle_{UT}\langle Proc^{(a)}, JoeP\rangle_{UT}$ $\langle K^{(b)}, JoeK\rangle_{UT}\langle t, film\rangle_{UT}\langle id_2, Ana\rangle_{UT}\langle id_3, Joe\rangle_{UT}\langle Proc^{(t)}, JoeP\rangle_{UT}\langle K^{(z)}, JoeK\rangle_{UT}$ $\langle\rangle_{UT}\ SK\ AK_1\ \langle\rangle_{AK}IPProc\ AProc_1\ \langle Parent\rangle$ where:
  $IPProc = \langle get\_answer(History : PairSet) :: (string)(string)\rangle_{AP}$ $\langle check\_adequacy(r : string) :: boolean\rangle_{AP}\langle pay(amount : real) :: null\rangle_{AP}$ is the string representation of the decision procedures of role IP.

Above we have shown the first five derivation steps of the Σ_{sell} system. In those derivation steps the following MAP^a operations have been simulated: initialization of the scene, invocation of a decision procedure, interchange of messages and creation of new instances due to change of role. Below we simulate how agents are inactivated in Σ_{sell} due to termination of their operation.

For the purpose of illustration of a dialogue termination let us consider the derivation step z when agents $A_{1,(Joe,1)}$ and $A_{3,(Joe,1)}$ are inactive, agent $A_{2,(Ana,1)}$ is suspended and agent $A_{4,(Ana,1)}$ has finished the execution of its operation and its active. Then we consider the Σ_{sell} system in the following configuration at derivation step z:

$$\sigma_{\Sigma_{sell},z} = (w_{E,z}, \{w_{1,(Joe,1)}\}, \{w_{2,(Ana,1)}\}, \{w_{3,(Joe,1)}\}, \{w_{4,(Ana,1)}\}) \text{ where:}$$

- $w_{E,z} = \varepsilon \ \langle\rangle_{NA,4,Ana}\langle\rangle_{M,4,Ana}\langle\rangle_{AA,4,Ana}$
- $w_{1,(Joe,1)} = \alpha_1 \ \langle null\rangle_{OP} \ \alpha_2 \ \langle Active\rangle \ \alpha_3$
- $w_{2,(Ana,1)} = Ana[\_,\_,\_] \downarrow \langle\rangle_{IM}\langle null\rangle_{OP}\langle Suspended\rangle\langle id_1, Ana\rangle_{UT}\langle t, film\rangle_{UT}$ $\langle id_1, Joe\rangle_{UT}\langle\rangle_{UT} \ SK \ AK_2\langle\rangle_{AK} \ AProc_2 \ \langle Parent\rangle$
- $w_{3,(Joe,1)} = \beta_1 \ \langle null\rangle_{OP} \ \beta_2 \ \langle Active\rangle \ \beta_3$
- $w_{4,(Ana,1)} = Ana[Ana, 2, 1] \downarrow \langle\rangle_{IM}\langle null\rangle_{OP}\langle Active\rangle NewUt_4 \ Know \ NewAProc_4\langle Parent\rangle$
- $Know =$ $\langle History = \{(kind, ?), (kind, 36), (brand, ?), (speed, ?), (speed?, ?), (place, ?), (place, out)\}\rangle_{AK}$ $\langle currency = euro\rangle_{AK}\langle\rangle_{AK}$

Then in derivation step $z+1$ agent $A_{4,(Ana,1)}$ becomes inactive. Rule $[\boldsymbol{\psi_{4,(Ana,1)}.6}]$ is applied and the following Σ_{sell} configuration is reached:

$$\sigma_{\Sigma_{sell},z+1} = (w_{E,z+1}, \{w_{1,(Joe,1)}\}, \{w_{2,(Ana,1)}\}, \{w_{3,(Joe,1)}\}, \{w_{4,(Ana,1)}\}) \text{ where:}$$

- $w_{E,z} = \varepsilon \ \langle\rangle_{NA,2,Ana}\langle\rangle_{M,2,Ana}\langle\rangle_{AA,2,Ana}$
- $w_{1,(Joe,1)}$ and $w_{3,(Joe,1)}$ do not change.
- $w_{2,(Ana,1)} = Ana[\_,\_,\_] \downarrow \langle\rangle_{IM}\langle null\rangle_{OP}\langle Suspended\rangle\langle id_1, Ana\rangle_{UT}\langle t, film\rangle_{UT}$ $\langle id_1, Joe\rangle_{UT}\langle\rangle_{UT} \ \ SK \ AK_2 \ \langle\rangle_{AK} \ AProc_2 \ \langle Parent\rangle$
- $w_{4,(Ana,1)} = \gamma_1 \ \langle null\rangle_{OP} \ \gamma_2 \ \langle Active\rangle PrivateK \ \gamma_3$

6. In derivation step $z+2$ the Σ_{sell} system makes agent $A_{2,(Ana,1)}$ active. Rule $[\boldsymbol{\varphi_{2,(Ana,1)}.3}]$ is applied and the following Σ_{sell} configuration is reached:

$$\sigma_{\Sigma_{sell},z+1} = (w_{E,z+1}, \{w_{1,(Joe,1)}\}, \{w_{2,(Ana,1)}\}, \{w_{3,(Joe,1)}\}, \{w_{4,(Ana,1)}\}) \text{ where:}$$

- $w_{E,z+2} = \varepsilon \ \langle\rangle_{NA,2,Ana}\langle\rangle_{M,2,Ana}\langle\rangle_{AA,2,Ana}$
- $w_{1,(Joe,1)}$, $w_{3,(Joe,1)}$ and $w_{4,(Ana,1)}$ do not change.
- $w_{2,(Ana,1)} = Ana[\_,\_,\_] \downarrow \langle\rangle_{IM}\langle null\rangle_{OP}\langle Active\rangle\langle id_1, Ana\rangle_{UT}\langle t, film\rangle_{UT}$ $\langle id_1, Joe\rangle_{UT}\langle\rangle_{UT} \ SK \ AK_2\langle\rangle_{AK} \ AProc_2 \ \langle Parent\rangle$

7. In a similar way in derivation step *z+4* agent $A_{2,(Ana,1)}$ becomes inactive in Σ_{sell} and the following final configuration is reached:

$$\sigma_{\Sigma_{sell},z+4} = (\varepsilon, \{w_{1,(Joe,1)}\}, \{w_{2,(Ana,1)}\}, \{w_{3,(Joe,1)}\}, \{w_{4,(Ana,1)}\})$$

   where:

   - $w_{1,(Joe,1)} = \alpha_1 \langle null \rangle_{OP} \alpha_2 \langle Active \rangle \alpha_3$
   - $w_{2,(Ana,1)} = \beta_1 \langle null \rangle_{OP} \beta_2 \langle Active \rangle \beta_3$
   - $w_{3,(Joe,1)} = \gamma_1 \langle null \rangle_{OP} \gamma_2 \langle Active \rangle \gamma_3$
   - $w_{4,(Ana,1)} = \xi_1 \langle null \rangle_{OP} \xi_2 \langle Active \rangle \xi_3$

4.4 Comparing *ConvEREG* systems with other grammatical frameworks for the specification of dialogue protocols

In chapter 1 we mentioned different attempts to model dialogues with grammars. Excluding the approaches introduced in [CVJLMV99], [JL00] and [AJR01], the rest of the dialogue systems mentioned in chapter 1 differ from our approach on the following:

- They were conceived for the practical purpose of modeling human-computer dialogues. While our approach corresponds to a formal framework that could be used for studying formal properties of human-like dialogue interactions, but also modeling human-computer and computer-computer dialogues.

- They have been implemented, and some of them are currently been used. Our framework is theoretical and it provides the formal semantic of MAP^a, an extension of an executable language for multi agent systems.

- They are used in limited dialogue contexts with a fix number of speakers, in general 2-party question-answering human-computer dialogues. The main reason for their dialogue context restriction is the high complexity associated to the tasks of speech recognition and speech synthesis, which can be considerably reduced when a limited context is considered. For instance if the dialogue is related to a flight booking only the vocabulary related to this topic needs to be considered. Our framework abstracts from speech recognition and synthesis problems, therefore no context restriction has been considered. Besides *ConvEREG* systems can specify arbitrary n-party dialogues, $n \geq 0$, where the behavior of agents is given by protocol strings in the MAP^a language. Also, in our model the number of speakers can dynamically change during conversation. For example, at time t there are x speakers, and at time $t + 1$ the number of participants can change to $y \neq x$ due to some conversant leaving the conversation, or new speakers joining the talk.

The first study of Grammar systems for modeling dialogues is presented in [CVJLMV99], where the notion of an *Eco-Rewriting* system is presented as a variant of *Eco-Grammar* system. Agents are seen as interlocutors and the environment is interpreted as the shared dialogue context. The environment can reflect what the interlocutors say, their knowledge, the topic of conversation, etc. The context and the state of the interlocutors is modified by action and evolution rules. In the Eco-Rewriting systems agents are replaced for "linguistic agents", more generic rewriting systems that can be, for example, a generative grammar, an insertion/deletion system, a contextual grammar, a grammar with context conditions, and many more.

In [JL00], a formal framework for modeling dialogues is introduced as a variant of Grammar system called *Conversational Grammar* system. Evidence is given for stating that the framework is suitable for modeling conversations. Mechanisms like turn-taking, adjacency pairs or closings can be simulated, and features like coordination, cooperation, interaction, dynamism, flexibility, emergence or coherence can be provided. But their approach remains in the theoretical ground. The practical adequacy of their framework has not been investigated. Our work can be seen as a continuation of [JL00] due to the similarity between EREG systems and *Conversational grammars*. While they do not provide any information of how to use their framework for practical purposes, in this chapter we give a concrete example of the use of our framework.

In [AJR01] a Grammar system for simulating goal-oriented dialogues is presented and its practical adequacy is exemplified. Their framework is based on *CD Grammar systems with memories*. CD Grammar systems with memories where introduced in [CVJJP94] as finite sets of semi-conditional grammars, each of them associated with a stack called memory. During the common derivation, the component grammars send messages to each other and they use their memories for storing/erasing messages (words) that they receive. The shared sentential form is modified based on both the contents of the memories and the context conditions being associated with the agent's productions.

In [AJR01], they exemplify their approach defining a 2-party dialogue protocol between a customer whose goal is to buy a film and a computer system whose goal is to discover the film that the buyer needs. We presented their example of dialogue protocol in section 4.3.3. But apart from the example no general definition of a CD Grammar systems with memories is given in [AJR01] to simulate arbitrary conversations between n speakers. Their work can be seen as a step ahead in the formal definition of a Grammar system for modeling dialogues. But as the authors pointed out, their approach is very much in progress and their conclusions and results are still tentative.

With respect to the framework presented in [AJR01], below we list some of its features and we mention how they differ with our framework:

- No concept of population. For this reason they can not model in a clear, reusable and modular way common shared behavior. In our framework the notion of population is crucial.

- Grammars $G_{i,a}$, $1 \leq i \leq n + 1$, are defined. One grammar per speaker, plus one grammar for modeling the conversational environment. In our model $m \geq 0$ populations $\mathcal{A}_i$ are defined, one per conversational role, plus the conversational environment E. A speaker is represented by an agent state $w_{i,a}$ in class state W_i.

- The number of active speakers in a conversation can dynamically vary, but it is bounded by the number n of grammars that define the CD Grammar system. In our framework the number of active speakers can vary during the conversation in an arbitrary and unbounded way.

- Agents are represented as "white boxes". For each agent its internal dynamics and its way of interacting with the environment is specified. We represent agents as "grey boxes". We only specify the agent's observable behaviors that is directly related with the interaction and we hide the rest of the agent's implementation details. Therefore, their framework is not suitable for modeling dialogues where agents show very complex internal behaviours or where agents cannot be fully knowable- as for instance in open systems.

- Agents are defined as semi-conditional grammars $G_{i,a} = (N_{i,a}, T_{i,a}, P_{i,a})$ with rewriting rules $P_{i,a}$ for describing their behavior. In our framework the behavior of an agent $A_{i,a}$ from population $\mathcal{A}_i$ is given by: a MAP^a language protocol description stored as a substring of agent state $w_{i,a}$, and mappings ψ_i and φ_i for selecting rewriting rules. The limitations of the use of rewriting rules for defining agent protocols are clear: they are statically defined during the definition of the Grammar system and they remain fixed during the whole simulation. Only fixed state-based conversational spaces can be simulated with fix rewriting rules. In contrast with this, describing protocols as processes stored in strings permits:

  - Specifying complex agent's behaviors using the operators *then* and *or* to compose actions.
  - Defining protocols in a modular and reusable way. The scene $Sell$ introduced in figure 4.3 is a good example of this. Two roles IS and IP are provided to define a generic information-seeking dialogue and two roles clerk and customer are used as particular types of IS and IP roles. In figure 4.4 we introduce the scene $Hospital$ to simulate the dialogue between a doctor and a patient. The doctor asks questions to the patient in order to know his symptoms and to diagnose his disease. To define the scene $Hospital$ we use the same roles IS and IP presented in 4.3 but this time we instantiate the information seeker as a doctor and the information provider as a patient. Therefore, different scenes can be defined reusing role definitions, and this is done by simply instantiating the generic roles with more specific role definitions.
  - Providing parametrization and recursion using agent invocation.
  - Dynamically changing agent behavior during run-time using operation variables: this enables to describe scenarios where it is impossible or impractical to define the protocol beforehand. For example, we can consider again the MAP^a protocols in figure 4.4. After some questions the doctor gives a diagnosis and informs the patient the corresponding treatment to follow, given by operation variable op. The number of treatments that the doctor can choose is very large and it depends on the particular dialogue, it can be: a diet, exercise, medication, more analysis and studies, consult other specialists, etc. For dialogue definitions like this one the use of operation variables is crucial.
  - Accessing and modifying shared knowledge. In our framework agents can access shared scene knowledge, role knowledge and private knowledge, but they only modify their own private knowledge.

The approaches introduced in [CVJLMV99], [JL00] and [AJR01] share with our framework the following features:

- Belong to information state theory, according to the dialogue classification introduced in chapter 1.
- Divide the information state in a common environment and private agent states.
- Be a Grammar system.
- Focus on the dialogue structure avoiding the problems of human speech recognition and synthesis.
- Be a formal theoretical model, without implementation.

```
agent(id_1, Patient, Proc^(a), K^(b), t) ::
id_2=choose_doctor(t) then
ask(t) ⟹ agent(id_2,IS) then
agent(id_1, IP, Proc^(a), K^(b)).

agent(id_1, IP, Proc^(t), K^(z)) ::
[ request(q) ⟸ agent(id_2, IS) then
  History = History ∪ {(newq =?)} then
  agent(id_1, IP, Proc^(t), K^(z)) ]
or
[ (q,a)=get_answer(History ∪ K) then
  History = History ∪ {(q = a)} then
  inform(q, a) ⟹ agent(id_2, IS) then
  agent(id_1, IP, Proc^(t), K^(z)) ]
or null
or
[ agent(id_1, IS, Proc^(t), K^(z)) then
  agent(id_1, IP, Proc^(t), K^(z)) ]
or
[ solution(prot) ⟸ agent(id_2, IS) then
  a=checkadequacy(prot) then
  History = History ∪ {(solution = prot)} then
  [ [ accept() ⟹ agent(id_2, IS)
      then prot ]
    or refuse() ⟹ agent(id_2, IS) ] ]   car
```

```
agent(id_1, Doctor, Proc^(c), K^(d)) ::
ask(t) ⇐ agent(id_1,Patient)
then agent(id_1, IS, Proc^(c), K^(d)).

agent(id_1, IS, Proc^(m), K^(s)) ::
[ newq=next_question(History ∪ K)then
  History = History ∪ {(newq =?)} then
  request(newq) ⟹ agent(id_2, IP) then
  agent(id_1, IS, Proc^(m), K^(s)) ]
or
[ inform(q, a) ⟸ agent(id_2, IP) then
  History = History ∪ {(q = a)} then
  agent(id_1, IS, Proc^(m), K^(s)) ]
or null
or
[ agent(id_1, IP, Proc^(m), K^(s)) then
  agent(id_1, IS, Proc^(m), K^(s)) ]
or
[ prot=final_answer(t, History ∪ K) then
  solution(prot) ⟹ agent(id_1, IP) then
  History = History ∪ {(solution = prot)} then
  [ accept() ⟸ agent (id_1, IP) or
    refuse() ⟸ agent(id_1, IP) ] ]
```

Figure 4.4: A MAP^a scene for a dialogue protocol in a hospital

4.5 Main features of framework *ConvEREG*

By introducing *ConvEREG* systems, we define a grammatical framework for the specification of dialogue protocols that proves to be:

- *Based on interaction and agent observable behavior.* The abstraction from agent implementation details that are not concerned with interaction simplifies the representation, analysis, design and verification of the dialogues.

- *Generic*. It allows the interaction between agents implemented in different languages (heterogeneous agents). It does not assume any fixed semantic for: the locutions agents use for message interchange, the technology used to deliver messages, the way agents modify shared scene knowledge, and the way agents perform rational processes.

- *Highly expressive*. The agents participate in the conversation concurrently. Their operations are sequentially composed and allow the implementing of nondeterministic choices, message interchange, parameterized invocations of new agent instances, functions calls, and iterations (defining a role that encapsulates the iterative process).

- *Flexible*. It allows the definition of n-party dialogues, with $n \geq 1$. The number of agents can dynamically vary during run-time, and there is no restriction over the maximal number of active agents in the dialogues.

- *Dynamic*. Agents are dynamic entities that are created during run-time by other active agents. Agents become inactive when they finish the execution of their associated protocol. Agents can modify their private knowledge, share their knowledge, introduce new agents into the conversations, and change their roles.

- *Emergent*. The dialogic space is not fixed, but emerges from the interaction. Agents follow a protocol description in a dialogue, but nevertheless they can behave in an unpredictable and autonomous way. They can choose what to do next according to the information state by the invocation of decision procedures. Besides the use of operation variables allows the agents to dynamically change their protocol definition during run-time. Operations are treated as first order objects allowing an agent to inform others what to do in a given situation. This feature provides agents with a way to react according to the current state of the dialogue. Due to this characteristic dialogue protocols specified using *ConvEREG* systems can be used for simulating dialogues where it is impossible or impractical to know beforehand the whole dialogic space.

- *Modular*. Three different structural and behavioral layers are defined: scenes, roles and agent instances. Each level is provided with a set of knowledge. The concepts of scene and role allow, in a modular and reusable way, the definition of pieces of common behavior and knowledge. When an agent joins a scene or adopts a role, it inherits the knowledge and behavior associated with that layer.

- *An improvement with respect to previous attempts from Formal Language Theory to simulate dialogues*. As was explained in section 4.4.

- *Provided with features characteristic of human conversations* . From [JL00] [CVJLMV99] we know that EREG systems inherit from Eco-Grammar systems the following features that according to [Cla96] and [SSJ74] are characteristic of human conversations:
  - *Copresence.* Participants share the same physical environment. In EREG systems this is given by the environmental state w_E.
  - *Emergent activity.* The content and development of a conversation is not fixed in advanced, but emerges through conversational acts. According to w_E, mapping φ_i selects a set of rewriting rules to change the sate of agent A_i. According to w_i, mapping ψ_i selects a set of rewriting rules to modify agent state and environment state.
  - *Audibility.* Participants can hear each other. Agents speak by mapping ψ_i rewriting symbols in the common environmental state w_E. Agents listen when they check the content of w_E by mapping φ_i.
  - *Instantaneity.* Participants perceive each other's actions at no significant delay. In EREG systems the delay is of one time unit. An agent speaks in time unit t_0 and the receiver listens the message at time $t_0 + 1$.
  - *Evanescence.* Medium is evanescent, it fades quickly. In our model, P_E deletes what agents introduce to the environmental in the previous time unit. Or if an agent does not want to pay attention to what other agent says, it hears the message in one time unit and deletes it (forgets it) in the next one.
  - *Simultaneity.* Participants can produce and receive instantly and simultaneously. Because mappings φ_i and ψ_i are applied in parallel, simultaneity is possible.
  - *Extemporariness.* Participants formulate and execute their actions extemporaneously, in real time. In EREG systems time is given by a common clock.

Copresence and audibility are part of what in pragmatics is called the *dialogue context*. According to [CVJLMV99], a context includes the linguistic code, the knowledge and cultural beliefs of the participants, the assumptions of what the participants know or suppose, social interaction principles that are universal or correspond to a certain culture, time and space context in which the conversational act takes place, etc. In human-like talks all the participants in the conversation can hear what is said, even if the message is addressed to another conversant. In our framework the dialogue context at time t is given by the environment state $w_{E,t}$. We can think of $L_E(\Sigma, \sigma_0)$ as a collection of dialogue contexts and call it the dialogue history.

Take, for example, the $\Sigma_{sell} \in ConvEREG_4$ system presented in section 4.3.3. It describes a dialogue that takes place in a public place, a shop, between a customer, Ana, and a clerk, Joe. In general speakers do not know another speaker's knowledge. But from communication speakers can acquire knowledge about behavioral characteristics of others, find out about the internal states of the other agents like feelings, attitudes, knowledge, etc. Dialogue context can also be used to compare agent's knowledge.

For instance if we consider Σ_{sell}, we can collect the locutions uttered by the agents in the systems $\Sigma_1, ..., \Sigma_m$ that correspond to scenes where the agent Joe played the role of a clerk with different customers with identifiers $b_1, ..., b_m$. From the agents' observable behaviors for each system $\Sigma_1, ..., \Sigma_m$ with initial configurations $\sigma_{1,0}, ..., \sigma_{m,0}$ we can determine, according to some criteria, if the agent Joe is a good seller. We can choose, for example, the following criteria: an agent with identifier a is a good clerk if it always guesses correctly what kind of film fulfils the requirements of the customer. Formally:

$$good\_seller(a) \leftrightarrow \left[\begin{array}{l} \forall i : 1 \leq i \leq m \wedge \\ \left(\begin{array}{l} \exists w_{E,t}, w_{E,q} \in L_E(\Sigma_i, \sigma_{i,0}) : t \geq 0 \wedge q > t \wedge \\ \langle solution(film_i)[a, IS][b_i, IP] \rangle_{M,IS,a} \text{ is a substring of } w_{E,t} \wedge \\ \langle accept()[b_i, IP][a, IS] \rangle_{M,IP,b_i} \text{ is a substring of } w_{E,q} \end{array} \right) \end{array} \right]$$

The film shop can have a policy that a clerk with identifier a is a good seller if it always offers, from all the films that fulfil the requirements of the customer, the most expensive one:

$$good\_seller(a) \leftrightarrow \left[\begin{array}{l} \forall i : 1 \leq i \leq m \wedge \\ \left(\begin{array}{l} \exists w_{E,t} \in L_E(\Sigma_i, \sigma_{i,0}) : t \geq 0 \wedge \langle solution(film_i)[a, IS][b_i, IP] \rangle_{M,IS,a} \\ \text{is a substring of } w_{E,t} \wedge \\ \Big(\forall film : similar(film, film_i) \rightarrow price(film) \leq price(film_i) \Big) \end{array} \right) \end{array} \right]$$

For example if we take again the Σ_{sell} system presented in section 4.3.3 we can compute the uttered locutions from systems $\Sigma_1, ..., \Sigma_m$ with initial configurations $\sigma_{1,0}, ..., \sigma_{m,0}$ corresponding to scenes where the agent Joe plays the role of a clerk with different customers with identifiers $b_1, ..., b_m$. We can also compute the uttered locutions in the systems $\Sigma_{m+1}, ..., \Sigma_{2m}$ with initial configurations $\sigma_{m+1,0}, ..., \sigma_{2m,0}$ corresponding to the scenes where other agent Luis plays the role of a clerk with the same customers with identifiers $b_{m+1}, ..., b_{2m}$. We cannot know whether Joe or Luis knows more about films, because according to MAP^a language agent knowledge is private and inaccessible. But we can compute some dialogue contexts and choose which agent performs better according to certain criteria. For example, we can consider that the best clerk is the one who is faster to propose the right film to the customer. For instance an agent can be faster because it has more experience or knowledge and it can ask less questions in order to guess the right film. Formally:

$$better\_seller(a, c) \leftrightarrow \left[\begin{array}{l} \forall i : 1 \leq i \leq m \wedge m + 1 \leq l \leq 2m \wedge \\ \left(\begin{array}{l} \exists w_{E,t}, w_{E,g} \in L_E(\Sigma_i, \sigma_{i,0}) : t \geq 0 \wedge g \geq t \wedge \\ \langle solution(film_1)[a, IS][b_i, IP] \rangle_{M,IS,a} \text{ is a substring of } w_{E,i,t} \wedge \\ \langle solution(film_2)[c, IS][b_l, IP] \rangle_{M,IS,c} \text{ is a substring of } w_{E,l,g} \end{array} \right) \end{array} \right]$$

The notion of dialogue context was introduced in Linguistics but it has already been considered in Dialogue Theory with names like agent observable behaviors [VO02], histories of communication [EBHM03] or traces of communication [WGS87]. The use of dialogue contexts in Dialogue Theory has shown to be crucial for the application of formal strategies of proof [WGS87] [MC81] [EBHM03].

4.6 Formal Properties

With theorem 10 we prove that the family $EConvEREG_n$, for all $n \geq 1$, has the same computational power as Turing Machines:

Theorem 10 *$EConvEREG_n = RE$ for all $n \geq 1$*

Proof. According to definition 30 in an arbitrary Grammar system $\Sigma_s \in EConvEREG_n$ mappings φ_i and ψ_i, for all $1 \leq i \leq n$, are computable functions, with erasing rewriting rules. So $EConvEREG_n \subseteq RE$.

From [CVD90] we know that the family of *Extended Eco-Grammar* systems with one active agent, λ-rules for action, and λ-rules for evolution, has the same expressive power as Turing Machines. Knowing that $EEG_1^\lambda = RE$ what remains to be understand is that $EEG_1^\lambda \subseteq EConvEREG_1$.

Given an arbitrary Grammar system $\Sigma \in EEG_1^\lambda$ with initial configuration $\sigma_0 = (w_{E,0}, w_{i,0})$ such that $\Sigma = (V_E, P_E, A_1)$ and $A_1 = (V_1, P_1, R_1, \varphi_1, \psi_1)$, from Σ we can define the following MAP^a scene $s = (AI, R, P, K, M)$ where:

- $AI = agent(a_1, r_1, \emptyset, \{ownstate = w_{i,0}\})$,
- $K = \{envstate = w_{E,0}\}$,
- $M = \emptyset$,
- $R = \{\langle r_1, Procs, \emptyset, \emptyset\rangle\}$, with
  $Procs = \left\{ \begin{array}{l} String :: Apply\varphi_1(x, String)(y, String), String :: Apply\psi_1(x, String)(y, String), \\ String :: ApplyP_E(x, String)(y, String) \end{array} \right\}$
  to simulate computable functions φ_1, ψ_1 and P_E.
- P is the unitary set corresponding to the protocol for role r_1:

```
$agent(a_1, r_1, Procs, Knows)$ ::
$oldw_1 := w_1$ then
$oldw_E := w_E$ then
$w_1 := Apply\varphi_1(w_E, w_1)$ then
$w_E := Apply\psi_1(w_E, oldw_1)$ then
$w_E := ApplyP_E(oldw_E, w_E)$ then
$K := \{envstate = w_E\}$ then
$agent(a_1, r_1, \emptyset, \{ownstate = w_i\})$.
```

From the way we have defined scene s and from definition 30 it is possible to construct a Grammar system $\Sigma_s \in EConvEREG_1$ such that $L_E(\Sigma, \sigma_0) = L_E(\Sigma_s, \sigma_0) \cap V_E^*$, for an arbitrary initial configuration σ_0.

■

Below we introduce two types of *ConvEREG* systems that we call *Conditional Eco-Grammar* (*CondEG*) systems and *regularly Controlled Reproductive Eco-Grammar (rCREG)* systems.

4.7 *CondEG* systems

In this section we introduce a type of *ConvEREG* system that we call *Conditional Eco-Grammar* (*CondEG*) system. We define *CondEG* systems as a variant of *Eco-Grammar* system, therefore, in *CondEG* the number of speakers is fixed during the conversation, although not all of them are necessarily active. Dialogue protocols in *CondEG* systems with at most $n \geq 1$ agents and condition $c \in \{b, s\}$ have these restrictions:

- Every agent A_i, $1 \leq i \leq n$, is provided with two private sets of decision procedures. The first set corresponds to a finite set of conditional rules of the type $(e, f : P)$ such that $e \in V_E^*$ is a permitting context, $f \in V_E^*$ is a forbidding context and (V_i, P) is a 0L component scheme. The second set corresponds to a finite set of conditional rules of the type $(g, h : R)$ such that $g \in V_i^*$ is a permitting context, $h \in V_i^*$ is a forbidding context and (V_E, R) is a 0L component scheme.

- Every agent A_i, $1 \leq i \leq n$, is provided with a computable procedure to determine if it wants to change its mental state (modify w_i) according to its first set of private decision procedures. If $c = b$, in each derivation step all the agents check if they have rules with the shape $(e, f : P)$ in their first set of decision procedures such that w_E contains the block of information e ($w_E = \alpha e \beta$) and w_E does not contain the block of data f ($w_E \neq \alpha f \beta$). In parallel all the agents who have a rule in the first set that can be applied execute it over w_i. If $c = s$, in each derivation step all the agents check if they have rules with the shape $(e, f : P)$ in their first set of decision procedures such that the information $e = e_0...e_s$ is scattered in $w_E(w_i = \alpha_1 e_1 \alpha_2 ... e_n \alpha_{n+1})$ and w_E does not contain the data $f = f_0...f_t$ scattered ($w_E \neq \beta_1 f_1 \beta_2 ... f_n \beta_{n+1}$). In parallel all the agents who have a rule in the first set that can be applied execute it over w_E. In this way we can simulate with *CondEG* systems how all the speakers simultaneously listen and change their own mental states according to the state of the conversation.

- Every agent A_i, $1 \leq i \leq n$, is provided with a computable procedure to determine if it wants to participate in the conversation (modify w_E) according to its second set of decision procedures. If $c = b$, in each derivation step from all the agents that have some rule with the shape $(g, h : R)$ in their second set of decision procedures, and w_i contains the block of information g ($w_i = \alpha g \beta$), and w_i does not contain the block of data h ($w_i \neq \alpha h \beta$), only one agent is non-deterministically chosen to apply the rule R from $(g, h : R)$ over w_E. If $c = s$, in each derivation step from all the agents that have rules with the shape $(g, h : R)$ in their first set of decision procedures such that the information $g = g_0...g_s$ is scattered in w_i, ($w_i = \alpha_1 g_1 \alpha_2 ... g_n \alpha_{n+1}$) and w_i does not contain the data $h = h_0...h_t$ scattered ($w_i \neq \beta_1 h_1 \beta_2 ... h_n \beta_{n+1}$), only one agent is non-deterministically chosen to apply the rules R over w_E. Imposing the restriction that per time unit only one agent can modify the shared knowledge w_E guaranties that in all the dialogue protocols specified using *CondEG* systems speakers take turns. Turn-taking is a feature characteristic of human talks that the dialogues defined using *ConvEREG* systems do not necessarily have.

4.7.1 Formal Definition

Definition 31 *By a Conditional Eco-Grammar system (CondEG) of degree $n \geq 1$ and condition $c \in \{b, s\}$ we mean an Eco-Grammar system $\Sigma = (E, A_1, ..., A_n)$ where:*

- *$E = V_E$, with V_E a finite alphabet;*
- *Agent A_i, $1 \leq i \leq n$, is defined as*
$$A_i = \begin{pmatrix} V_i, \{(g, h : R) \mid g \in V_i^* \wedge h \in V_i^* \wedge (V_E, R) \textit{ is a } 0L \textit{ component scheme } \}, \\ \{(e, f : P) \mid e \in V_E^* \wedge f \in V_E^* \wedge (V_i, P) \textit{ is a } 0L \textit{ component scheme } \} \end{pmatrix}$$

In order to describe the dynamic aspects of *CondEG* systems we give the following definitions:

Definition 32 *(System configuration) A configuration of a CondEG system $\Sigma = (E, A_1, ..., A_n)$ of degree $n \geq 1$ with condition $c \in \{b, s\}$ is an (n+1)-tuple: $\sigma = (w_E, w_1, w_2, ..., w_n)$ where $w_E \in V_E^*$ and $w_i \in V_i^+$ for $1 \leq i \leq n$; w_E is the current evolution state of the environment and the string w_i corresponds to the evolution states of the active agent A_i.*

We introduce the definition of agent derivation interpreting the conditional predicate π_c, $c \in \{b, s\}$, as explained in Definition 33.

Definition 33 *Given an alphabet V, we define the following predicates over $V^* \times V^*$:*
$$\begin{array}{lll} \pi_b(x, y) = 1 & iff & y = y_1 x y_2, \\ \pi_s(x, y) = 1 & iff & y = y_1 x_1 y_2 x_2 ... y_r x_r y_{r+1}, \\ & & x = x_1 x_2 ... x_r \textit{ where } x_i, y_i \in V^* \textit{ for all } i. \end{array}$$

Definition 34 *(Agent Derivation) Agent state w_i derives to a new state w_i' denoted by $w_i \vdash w_i'$ if there is a rule $(g, h : R)$ in agent A_i such that $\pi_c(g, w_E) = 1$, $\pi_c(h, w_E) = 0$. Then $w_i \vdash w_i'$ is obtained as the result of the application of the 0L component scheme (V_i, R). The agent derivation is a computable process.*

Definition 35 *(Environment Derivation) Environmental state w_E derives to new state w_E' denoted by $w_E \models w_E'$ iff there is a rule $(d, e : P)$ corresponding to some active agent A_i such that $\pi_c(d, w_i) = 1$, $\pi_c(e, w_i) = 0$ and $w_E \models_P w_E'$ is the result of the application of the 0L component scheme (V_E, P). The environment derivation is a computable process.*

Definition 36 *(System derivation) Let $\Sigma = (E, A_1, ..., A_n)$ be a CondEG system with condition $c \in \{b, s\}$ and let $\sigma_t = (w_E, w_1, w_2, ..., w_n)$ and $\sigma_{t+1}' = (w_E', w_1', w_2', ..., w_n')$ be two configurations of Σ. We say that σ_t is directly changed for (it directly derives) σ_{t+1}', denoted by $\sigma_t \Longrightarrow_\Sigma \sigma_{t+1}'$, iff w_E' arises from w_E by evolution of the environment produced by some active agent in σ_t, and w_i' is the result of the agent derivation over w_i, $1 \leq i \leq n$.*

We denote as $\Rightarrow_\Sigma^+$ and $\Rightarrow_\Sigma^*$ the transitive and reflexive closure of $\Longrightarrow$ in Σ.

Definition 37 *(System language) The language generated by $\Sigma \in$ CondEG with condition $c \in \{b, s\}$ and initial configuration σ_0 is defined as*

$$L_E(\Sigma, \sigma_0) = \left\{ \begin{array}{c} w_E \mid \sigma_j = (w_E, w_1, ..., w_n), \\ \sigma_0 \Longrightarrow_\Sigma \sigma_1 \Longrightarrow_\Sigma .... \Longrightarrow_\Sigma \sigma_j, j \geq 0 \end{array} \right\}$$

We denote by $CondEG_n(i, j, c)$, $n \geq 1$, $c \in \{b, s\}$, $i, j \geq 0$ the family of languages $L_E(\Sigma, \sigma_0)$ where $\Sigma \in CondEG$ has degree n, initial configuration σ_0, all the permitting contexts have length at most i, all the forbidden contexts have length at most j and predicate π_c is used to verify the presence or absence of contexts. When the number of agents, the length of the permitting context, or the length of the forbidding contexts is not bounded, we replace the corresponding parameter with ∞.

When we are interested only in strings over some alphabet T, hence in the language $L_E(\Sigma, \sigma_0) \cap T^*$, we speak about an *Extended Conditional Eco-Grammar (ECondEG)* system. We denote by $ECondEG_n(i, j, c)$, $n \geq 1$, $c \in \{b, s\}$, $i, j \geq 0$, the family of languages $L_E(\Sigma, \sigma_0) \cap T^*$ where $\Sigma \in CondEG_n(i, j, c)$.

4.7.2 Formal Properties

The following basic relations directly follow from the definition of *ECondEG* system:

Lemma 1

1. $ECondEG_n(i, j, c) \subseteq ECondEG_m(i', j', c)$, *for all* $c \in \{b, s\}$, $1 \leq n \leq m$ *and* $i \leq i'$, $j \leq j'$.
2. $ECondEG_n(i, j, b) \subseteq ECondEG_n(i, j, s)$, *for all* $n \geq 1$ *and* $i, j \in \{0, 1\}$.
3. $ECondEG_n(0, 0, c) = ET0L$, *for all* $n \geq 1$, $c \in \{b, s\}$.

The result below connects *ECondEG* systems with the *Extended Conditional Tabled Eco-Grammar (ECTEG)* systems introduced in chapter 2.

Theorem 11 $ECTEG_n(i, j, c) \subseteq ECondEG_1(i, j, c)$, *for all* $i, j \geq 0$, $n \geq 1$, $c \in \{b, s\}$

Proof. In [Mih94], it was proved that $ECTEG_n(i, j, c) = ECTEG_1(i, j, c)$ for arbitrary $n \geq 1$, $i, j \geq 0$ and $c \in \{b, s\}$. Thus we can prove this theorem replacing n for 1. Given an arbitrary $W \in ECTEG_1(i, j, s)$ such that $W = (E, A_1)$ with:

- $E = (V_E, Rules_E)$,
- $Rules_E = \{(e_1, f_1 : P_1), ..., (e_k, f_k : P_k)\}$,
- $A_1 = (V_1, Rules_1)$,
- $Rules_1 = \{(g_1, h_1 : R_1), ..., (g_s, h_s : R_s)\}$,
- $V_E \cap V_1 = \emptyset$.

From W we can define $W' \in ECondEG_1(i, j, c)$ such that $W' = (V_E, A'_1)$ with $A_1 = (V_1, Rules_E, Rules_1)$. Clearly for an arbitrary initial configuration σ_0 it is true that $L_E(W, \sigma_0) = L_E(W', \sigma_0)$. ■

We can prove now the connection between *ECondEG* systems with non erasing rules and the Chomsky hierarchy:

Theorem 12 *$ECondEG_{e,n}(i, j, c) \subseteq CS$, for all $n \geq 1$, $i, j \geq 0$, $c \in \{b, s\}$.*

Proof.

To prove that $ECondEG_{e,n}(i, j, c) \subseteq CS$ we prove first that $ECondEG_{e,n}(i, j, c) \subseteq EG_e(1)$ where $EG_e(1)$ is the family of *Extended Eco-grammar systems* with non-erasing rules and 1 agent, because according to [CVKKP97] $EG_e(1) = CS$.

From an arbitrary $\Sigma \in ECondEG_{e,n}(i, j, c)$ such that $\Sigma = (E, A_1, ..., A_n)$ we can always construct $\Sigma' \in ECondEG_{e,n}(i, j, c)$ that only differs from Σ on the alphabets V_i used by the agents $A_1, ..., A_n$ which are disjoint.

We consider $\Sigma' = (V_E, A_1, ..., A_n)$ such that for all $1 \leq i \leq n$: $A_i = (V_i, R_{1,i}, R_{2,i})$. From Σ' we can define an Eco-grammar system $\Gamma \in EG_e(1)$ such that $\Gamma = ((V_E, \emptyset), B_1)$ where: $B_1 = ((V_1 \cup ... \cup V_n), (R_{1,1} \cup ... \cup R_{1,n}), (R_{2,1} \cup ... \cup R_{2,n}), \varphi_1, \psi_1)$ and functions φ_1 and ψ_1 simulate the derivations that can be performed in Σ' such that $L_E(\Sigma') = L_E(\Gamma)$.

■

4.7.3 Applicability in Dialogue Theory

Family $CondEG_n$ provides a formal framework for specifying dialogue protocols with the following features:

- The number of speakers is fixed to n.
- The agents can learn or modify their own knowledge bases and the shared knowledge base through the dialogue.
- Any dialogue initiative, backtracking, turn-taking and reply policy can be implemented.
- There is no restriction over the representation of the agent's mental states or shared knowledge, which correspond to strings over two potentially different finite alphabets.
- The knowledge base of an agent is restricted to two finite sets of conditional rules. One of these sets is used by the agent to decide how to change its mental state according to the absence and presence of information in the dialogue context. The other set is used by the agent to decide what to say according to the absence and presence of information in its mental state.
- The complexity of the agent reasoning strategies is limited to find a conditional rule from its knowledge base that can be applied, in the sense of definition 36, over the shared knowledge.

4.8 *rCREG* systems

We introduce a type of *ConvEREG* system that we call *regularly Controlled Reproductive Eco-Grammar (rCREG)* system. We define *rCREG* systems as *Reproductive Eco-Grammar (REG)* systems, where a policy of turns is specified by a regular set X over $\mathcal{A}_1, ..., \mathcal{A}_n$. If according to

$L(X)$ it is the turn of an arbitrary agent from population $\mathcal{A}_i$ to talk, then an agent $A_{j,i}$ in $\mathcal{A}_i$ is non-deterministically chosen to change his mental state and participate in the conversation by applying an arbitrary λ-free CF rule over w_E.

Dialogue protocols defined using *rCREG* systems mainly differ from protocols specified using *ConvEREG* systems on the following:

- Agents are provided with a turn talking policy, given by a regular set X over populations $\mathcal{A}_1, ..., \mathcal{A}_n$, so that agents cannot speak simultaneously or in an arbitrary order.
- The behavior of the speakers is given by λ-free rewriting rules, and not by protocol descriptions in the MAP^a language.
- The participation of each agent A_i in the dialogue is fixed and it corresponds to the application of an arbitrary rewriting rule from the finite set of λ-free CF rules R_i over the environment state. Therefore, the mental states of the agents do not influence agents participation in the dialogue.
- The set of environment rules P_E is empty.

4.8.1 Formal Definition

Definition 38

$$rCREG(X) = \left\{ \begin{array}{l} \Sigma \in REG \mid \Sigma = (E, \mathcal{A}_1, ..., \mathcal{A}_n) \wedge E = (N \cup T, \emptyset) \wedge \\ \left(\begin{array}{l} \forall 1 \leq i \leq n : A_i = (V_i \cup \{\sqcup\}, P_i, R_i, \varphi_i, \psi_i) \wedge R_i : N \to (N \cup T)^+ \wedge \\ \psi_i \textit{ returns an arbitrary production in } R_i \end{array} \right) \wedge \\ L_E(\Sigma, \sigma_0) = \left\{ \begin{array}{l} w_{E,j} \in N \cup T^* \mid j \geq 0 \wedge \sigma_j = (w_{E,j}, W_{1,j}, ..., W_{n,j}) \wedge \\ w_{E,0} \in N \wedge \sigma_0 \Longrightarrow_{\Sigma}^{p_{y_1,x_1}} \sigma_1 \Longrightarrow_{\Sigma} ... \Longrightarrow_{\Sigma}^{p_{y_j,x_j}} \sigma_j \wedge \\ \left(\begin{array}{l} \forall 1 \leq k \leq j : w_{E,k-1} \models_1^{p_{y_k,x_k}} w_{E,k} \wedge p_{y_k,x_k} \in R_{x_k} \wedge \\ \mathcal{A}_{x_1}, ..., \mathcal{A}_{x_j} \in L(X) \end{array} \right) \end{array} \right\} \end{array} \right\},$$

for all X regular set over populations $\mathcal{A}_1, ..., \mathcal{A}_n$.

We denote by $rCREG_n(X)$, $n \geq 1$, X a regular expression over n populations, the family of languages $L_E(\Sigma, \sigma_0)$ where $\Sigma \in rCREG(X)$ has degree n and initial configuration σ_0. When the number of populations is not bounded, then we replace the corresponding parameter with ∞.

When we are interested only in strings over some alphabet $T \subseteq E$, hence in the language $L_E(\Sigma, \sigma_0) \cap T^*$, then we speak about an *Extended regularly Controlled Reproductive Eco-Grammar (ErCREG)* system. We denote by $ErCREG_n(X)$, the family of languages $L_E(\Sigma, \sigma_0) \cap T^*$ where $\Sigma \in rCREG_n(X)$.

4.8.2 An example of dialogue protocol

We consider $P \in rCREG_3(\mathcal{A}_1(\mathcal{A}_1\mathcal{A}_2)^*\mathcal{A}_3\mathcal{A}_3)$ such that $P = (E, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ with:

- Initial configuration $\sigma_0 = (S, \{S_1\}, \{S_2\}, \{S_3\})$,
- $E = (V_E, \emptyset)$,

- $V_E = N \cup T = \{S, A, B\} \cup \{a, b, c\}$,
- for all $1 \leq i \leq 3$, $\mathcal{A}_i = (V_i \cup \{\sqcup\}, P_i, R_i, \varphi_i, \psi_i)$, $V_i = \{S_i\}$, $P_i = \{S_i \rightarrow S_i\}$, $\varphi_i(w_E) = P_i$ for all $w_E \in V_E^*$,
- $R_1 = \{S \rightarrow AB, A \rightarrow aAb\}$, $R_2 = \{B \rightarrow cB\}$, $R_3 = \{A \rightarrow ab, B \rightarrow c\}$.

With the regular set $(\mathcal{A}_1(\mathcal{A}_1\mathcal{A}_2)^n\mathcal{A}_3\mathcal{A}_3)$, $n \geq 0$, the only possible derivation is the following:

$(S, \{S_1\}, \{S_2\}, \{S_3\}) \Rightarrow (AB, \{S_1\}, \{S_2\}, \{S_3\}) \Rightarrow^n (a^nAb^nc^nC, \{S_1\}, \{S_2\}, \{S_3\}) \Rightarrow^2$
$(a^{n+1}b^{n+1}c^{n+1}, \{S_1\}, \{S_2\}, \{S_3\})$

Therefore, $L_E(P, \gamma_0) = \{a^nb^nc^n \mid n \geq 0\}$, $L_E(P, \gamma_0) \in CS$.

4.8.3 Formal Properties

In [GS68] [DPS97], *regularly Controlled* (context free) grammars are formally introduced as a type of mildly context sensitive grammar. *Regularly controlled* (context free) grammars are basically context free grammars provided with a sequence of productions, and we only take those words in the generated language that can be obtained by context free derivation using productions according to defined sequences. By *rC* we denote the family of languages generated by *regularly Controlled* grammars without erasing rules.

Theorem 13 *$ErCREG_n(X) = rC \subset CS$, for all $n \geq 1$, regular set X over n populations.*

Proof. In [DPS97] it was proved that $rC \subset CS$.

First we prove that $ErCREG_n(X) \subseteq rC$.

Given an arbitrary dialogue protocol $M \in ErCREG_n(X)$ such that $M = (E, , \mathcal{A}_1, ..., \mathcal{A}_n)$ with:

- $E = (N \cup T, \emptyset)$ and
- $\sigma_0 = (w_{E,0}, W_{1,0}, ..., W_{n,0})$.

From M we define $G_M \in rC$, $G_M = (N, T, P, w_{E,0}, Q)$ such that:

- $P = \{p_{j,\mathcal{A}_i} \mid 1 \leq i \leq n \wedge p_j \in R_i\}$,
- Q is the regular expression resulting from replacing in X each occurrence of population $\mathcal{A}_i$, $1 \leq i \leq n$, for expression $(p_{1,\mathcal{A}_i} + ... + p_{t,\mathcal{A}_i})$ such that $R_i = \{p_1, ..., p_t\}$.

We need to prove the following: $(\forall \alpha : \alpha \in L_E(M, \sigma_0) \cap T^* \leftrightarrow \alpha \in L(G_M))$.

Clearly it is satisfied that:

$$\left(\forall j \geq 0 : \left(\begin{array}{r} \sigma_j = (w_{E,j}, W_{1,j}, ..., W_{n,j}) \wedge w_{E,j} \in T^* \wedge \sigma_0 \Longrightarrow_M \sigma_1 \Longrightarrow_M ... \Longrightarrow_M \sigma_j \wedge \\ \left(\forall 1 \leq k \leq j : w_{E,k-1} \models_1^{p_{y_k,x_k}} w_{E,k} \wedge p_{y_k,x_k} \in R_{x_k} \wedge \mathcal{A}_{x_1}...\mathcal{A}_{x_j} \in L(X) \right) \leftrightarrow \\ \left(\begin{array}{l} w_{E,0} \overset{p_{y_1,\mathcal{A}_{x_1}}}{\Longrightarrow}_{G_M} w_{E,1} \overset{p_{y_2,\mathcal{A}_{x_2}}}{\Longrightarrow}_{G_M} ... \overset{p_{y_j,\mathcal{A}_{x_j}}}{\Longrightarrow}_{G_M} w_{E,j} \wedge \\ p_{y_1,\mathcal{A}_{x_1}}...p_{y_j,\mathcal{A}_{x_j}} \in L(Q) \end{array} \right) \end{array} \right) \right).$$

Therefore $L_E(M, \sigma_0) \cap T^* = L(G_M)$.

Then we prove that $rC \subseteq ErCREG_n(X)$.

Given an arbitrary grammar $G \in rC$ such that $G = (N, T, P, S, Q)$ and $P = \{p_1, ..., p_n\}$, $n \geq 1$. From G we define $M \in ErCREG_n(X)$ such that:

- $M = ((N \cup T, \emptyset), \mathcal{A}_1, ..., \mathcal{A}_n)$;
- $\sigma_0 = (S, \{S\}, ..., \{S\})$;
- Agent $A_{1,j}$ in every class $\mathcal{A}_j$, $1 \leq j \leq n$, is defined as $A_{1,j} = (\{S\}, R_j, \emptyset, \varphi_j, \psi_j)$ with $R_j = \{p_j \mid p_j \in P\}$ and $card(R_j) = 1$;
- X results from replacing in Q the occurrence of each production $p_j \in P$ for population $\mathcal{A}_j$, $1 \leq j \leq n$.

We need to prove the following: $(\forall \alpha : \alpha \in L(G) \leftrightarrow \alpha \in L_E(M, \sigma_0) \cap T^*)$.

Clearly it is satisfied that:

$$\left(\begin{array}{l} \forall j \geq 0 : \left(\; w_0 \overset{p_{x_1}}{\Longrightarrow}_G w_1 \Longrightarrow_G ... \overset{p_{x_j}}{\Longrightarrow}_G w_j \wedge p_{x_1} ... p_{x_j} \in L(Q) \wedge w_j \in T^* \; \right) \leftrightarrow \\ \qquad\qquad \left(\begin{array}{l} \sigma_j = (w_j, \{S\}, , ..., \{S\}) \wedge \sigma_0 \Longrightarrow_M \sigma_1 \Longrightarrow_M ... \Longrightarrow_M \sigma_j \wedge \\ (\forall 1 \leq k \leq j : w_{k-1} \models_1^{p_{x_k}} w_k \wedge p_{x_k} \in R_{x_k}) \wedge \mathcal{A}_{x_1} ... \mathcal{A}_{x_j} \in L(X) \end{array} \right) \end{array} \right).$$

Then $L(G) = L_E(M, \sigma_0) \cap T^*$. ■

From theorem 13 and the results from [DPS97] which state that for the *rC* grammars the membership problem is decidable, the emptiness problem is NP-hard and the finiteness problem is NP-hard, we get the following corollary:

Corollary 3 *In family $ErCREG_n(X)$, for all $n \geq 1$, for all regular set X over n populations, the membership problem is decidable, the emptiness problem is NP-hard and the finiteness problem is NP-hard.*

From theorem 10 and theorem 13 we deduce:

Corollary 4 *$ErCREG_n(X) \subset EConvEREG_m$, for all $n, m \geq 1$ and X regular set of agent populations.*

4.8.4 Applicability in Dialogue Theory

rCREG systems provide a formal framework to specify dialogue protocols with the following features:

- There are no limits on the maximum number of speakers.
- The agents can learn and modify their own knowledge bases and the shared knowledge base through the dialogue.

- Backtracking, turn-taking and different replying policies can be implemented.
- There is no restriction over the representation of agent's mental state and shared knowledge.
- Any locution set and dialogue initiative can be chosen.
- The way agents participate in the dialogue is not influenced by the agent's mental state and it is limited to the application of an arbitrary CF rule to modify the string of shared knowledge.

Chapter 5

Using finite state transition systems for specifying dialogue protocols

5.1 Introduction

In chapter 4 we introduced the *Conversational Enlarged Reproductive Eco-Grammar* (*ConvEREG*) systems as a variant of Reproductive Eco-Grammar system. The grammar components in *ConvEREG* are interpreted as speakers whose behaviors are specified by strings corresponding to process descriptions in the *Multi-Agent Protocol* (MAP^a) language that we introduced in [GW06b] [GW06c] [GW06a]. We start this chapter defining a framework for the specification of dialogue protocols that corresponds to a variant of finite state transition systems, that we call *Conversational Finite State Transition (ConvFST)* systems. $ConvFST$ systems have the same expressive power as the framework *ConvEREG*. This approach can be seen as a continuation of similar formal models based on finite state automata for the simulation of dialogues, like [Vas04, ERS$^+$01] [HK98][MW97][PC96].

Finite state transition systems have been popular mechanisms to describe frameworks for the definition of dialogue protocols because they are considered to have the following features:

- They are easy and understandable methods for describing protocols.
- They are provided with a formal semantics.
- They can be subject of a verification strategy called model checking.
- They allow to specify dialogues where speakers take turns.

$ConvFST$ systems allow the specification of dialogue protocols where the number of agents is fixed, agents are provided with fix sets of private believes, the interaction model is described by a finite state transition system whose transitions are conditionals and labeled with locutions, and the shared knowledge is saved in a string whose content can be modified. The states of the automata correspond to possible stages of the conversation, and the transitions to dialogue moves. For a labeled transition to be triggered the precondition associated with the locution that labels it has to be satisfied and checking its satisfaction is computable. And the preconditions of the locutions are formulas over some logic which are evaluated according to the agent knowledge and shared knowledge. If a transition is triggered a new state is reached and some operation can be performed over the shared knowledge. $ConvFST$ systems correspond to dialogue state approaches.

In sections 5.2.1 and 5.2.2 respectively, we define $ConvFST$ systems and we provide an example of the use of this framework defining an information-seeking argumentation-based dialogue protocol that we call $\mathcal{ISP}$. With $\mathcal{ISP}$ we explain that once a dialogue protocol is defined in a framework like $ConvFST$ formal properties can be expressed and proved. In section 5.2.3 we make a brief introduction to a subfield of Dialogue Theory called Argumentation Theory. Argumentation Theory is based on the interchange of arguments and contraarguments between speakers, with the purpose of arriving to conclusions even in the presence of incomplete or inconsistent information. In section 5.2.3 we provide an example that shows that $ConvFST$ systems can be used as a transition-based formal framework for the definition of protocols defined in the argumentation-based formal framework introduced by Amgoud in [Amg98].

In section 5.2.4 we prove some formal properties for $ConvFST$ systems. With theorem 15 we specify under which restrictions the family $ConvFST_n$ of $ConvFST$ systems with at most $n \geq 1$ agents collapses into the family $ConvFST_1$ of $ConvFST$ systems with at most 1 agent. With theorem 16 we prove that the expressive power of $ConvFST$ systems of any degree is equal to the expressive power of Turing Machines.

For $ConvFST$ systems the membership problem is undecidable, therefore we define three subclasses of $ConvFST$ with less expressive power: *Conditional Finite State Transition* ($CondFST$) systems, *limited memory Finite State Transition (lmFST)* systems and *regularly Controlled Finite State Transition (rCFST)* systems.

In section 5.3 we define $CondFST$ systems as $ConvFST$ systems where the agents are provided with a finite set of conditional rules that they use to decide their participation in the dialogue. The agents use these rules to decide how to modify the string corresponding to the shared knowledge depending on the presence and\or absence of substrings (information) in that string. With theorem 17 we prove that the family $ECondFST_n$ of *Extended* $CondFST$ systems with at most $n \geq 1$ agents collapses into the family $ECondFST_1$ of $ECondFST$ systems with at most 1 agent. Theorem 18 proves that the family $ECTEG_n(i, j, c)$ of classes of languages defined by *Extended Conditional-Tabled Eco-Grammar systems* with $n \geq 1$ agents, forbidding and permitting context of length at most $i, j \geq 0$ and with block or scattered condition $c \in \{b, s\}$ is included in the family $ECondFST_1(\infty, \infty, c)$. Where $ECondFST_1(\infty, \infty, c)$ is the family of classes of languages generated by $ECondFST$ systems with at most $n \geq 1$ agents, permitting contexts of length at most $i \geq 0$, forbidding contexts of at most $j \geq 0$, conditions $c \in \{b, s\}$ blocked or scattered. Theorem 19 proves that $ECondFST$ systems without erasing rules are a subclass of CS grammars.

In section 5.4 and in [Gra08] we introduce $lmFST$ systems as $ConvFST$ systems where the string of shared knowledge is replaced for a memory limited to at most the last uttered locution. Because there are no restrictions over the locutions used very complex dialogues can be modeled, where locutions can contain information of previously uttered locutions and even the whole dialogue history. Then we provide an example that shows the possible implications that considering a memory with this restriction, instead of a string of locutions, can have over the specification in $lmFST$ of argumentation-based dialogues. In section 5.4 we prove some formal properties for $lmFST$ systems. With theorem 20 we prove that the family $lmFST_n$ of $lmFST$ systems with at most $n \geq 1$ active agents collapses into the family $lmFST_1$. With theorem 21 we prove that the family $ElmFST_n$ of *Extended limited memory Finite State Transition (ElmFST)* systems with arbitrary $n \geq 1$ agents has the same expressive power as CS grammars.

In section 5.5 we introduce *rCFST* systems as *ConvFST* systems where the locutions have no parameters and each locution is associated with a λ-free CF rule such that the locution's precondition evaluates true iff the associated CF rule can be applied over the string of shared knowledge. If the precondition associated with a locution evaluates true, then the locution is uttered and a new string of shared knowledge is obtained from the application of the CF rewriting rule associated with that locution. After explaining the suitability of this framework to specify frame-based mixed-initiative dialogue protocols, we prove in section 5.5.3 some of its formal properties. With theorem 22 we prove that the family of *Extended rCFST* systems with at most $n \geq 1$ agents ($ErCFST_n$) is equivalent to the family *rC* of languages generated by *regularly Controlled* grammars with λ-free CF productions [GS68] [DPS97]. Therefore $ErCFST_n \subset CS$ and the problems of membership, emptiness and finiteness are decidable, with the last two problems being NP-hard. With theorem 23 we prove how the family $rCFST_n$ collapses with the family $rCFST_1$ when we consider the language of shared knowledge generated.

The overall intention of chapters 4 and 5 is to contribute to Dialogue Theory, mainly from the perspective of Formal Language Theory, in the formal study of frameworks for the specification of dialogues. In this chapter we enlarge the hierarchy of Grammar system variants presented in chapter 4 which frameworks based on finite state transition systems for the specification of dialogue protocols.

5.2 *ConvFST* systems

In *Conversational Finite State Transition* (*ConvFST*) systems the number of agents is fixed. Each agent A_i, $1 \leq i \leq n$, is provided with a private knowledge base K_i containing its beliefs. These knowledge bases are fixed during the dialogue, which means that the agents do not learn through the dialogue. The dialogue moves are described by a finite state transition system where the states correspond to stages in the conversation and transitions are conditional and labeled by locutions from a locution set. For a labeled transition to be triggered the precondition associated with the locution that labels it has to be satisfied. The preconditions of the locutions are formulas over some logic which are evaluated according to the agent knowledge and shared knowledge. Checking the satisfaction of locution's precondition is computable. If a transition is triggered a new state is reached and some operation can be performed over a string of shared knowledge. This string of shared knowledge records the information exchanged and knowledge learned by the agents during the dialogue.

ConvFST systems allow the definition of dialogue protocols with: backtracking (the capacity to reply to locutions uttered at any earlier step of the dialogue and not only the previous one), different turn-taking rules (agents can make several moves before the turns shift), arbitrary sets of locutions, arbitrary types of agent knowledge bases and reasoning strategies and multiple replies (indeterministic transition systems).

5.2.1 Formal Definition

Definition 39 *A dialogue protocol $W \in ConvFST$ of degree $n \geq 1$ is a tuple:*

$$W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{L}}, \Gamma, \delta, q_0, SK_0, F)$$

where:

- *Σ is a finite set of symbols;*
- *$K_{id_i} \in 2^{\Sigma^+}$, for all $1 \leq i \leq n$, are sets of strings over alphabet Σ. We call to K_{id_i} the private knowledge base of the agent with identifier id_i;*
- *Q is a finite set of states;*
- *$LS_{\mathcal{L}}$ is a finite set of locutions,*
  $LS_{\mathcal{L}} = \left\{ \rho_{id_i}(\phi^{(m)}) \mid m \geq 0 \wedge 1 \leq i \leq n \wedge prec(\rho_{id_i}(\phi^{(m)})) \text{ is a wff in logic } \mathcal{L}\} \right\}$
  The locution $\rho_{id_i}(\phi^{(m)}) \in LS_{\mathcal{L}}$ with constants ρ, id_i and terms $\phi^{(1)}, ..., \phi^{(m)}$ has associated a well formulated formula $prec(\rho_{id_i}(\phi^{(m)}))$ in logic $\mathcal{L}$. We call to $prec(\rho_{id_i}(\phi^{(m)}))$ the precondition of locution $\rho_{id_i}(\phi^{(m)})$, which can be evaluated using knowledge from K_{id_i} and SK. Checking a locution's precondition is a computable function. There cannot be a locution $\rho_{id_i}(\phi^{(m)}) \in LS_{\mathcal{L}}$ with more than one precondition;
- *Γ is a finite set of symbols used for the representation of the string $SK \in \Gamma^*$ corresponding to the dialogue shared knowledge;*
- *$SK_0 \in \Gamma^*$ is the string corresponding to the initial shared knowledge;*
- *$q_0 \in Q$ is the initial state;*
- *$F \subseteq Q$, are the final states, and*
- *δ is a finite transition relation $(Q \times LS_{\mathcal{L}} \times \Gamma^*) \rightarrow 2^{Q \times \Gamma^*}$.*

Considering that a sequence of locutions $\rho_{id}(v^{(m)})\beta \in (LS_{\mathcal{L}})^+$ is being processed, for any $q \in Q$, $\rho_{id}(v^{(m)}) \in LS_{\mathcal{L}}$, with parameter values $v^{(m)}$, $\beta \in (LS_{\mathcal{L}})^*$, $SK \in \Gamma^*$, the interpretation of

$$\delta(q, \rho_{id}(v^{(m)}), SK) = \{(p_1, newSK_1), ..., (p_m, newSK_m)\}$$

is that if the dialogue protocol W is in state q with current locution $\rho_{id}(v^{(m)}) \in LS_{\mathcal{L}}$, with shared knowledge SK, and the formula $prec(\rho_{id}(v^{(m)}))$ evaluates true, then W can for any $1 \leq j \leq m$ replace SK with $newSK_j$, take the first locution in β and enter state p_j.

To formally describe the configuration of a dialogue protocol W at a given instant we define what we call an *instantaneous description*. An instantaneous description records the state, the sequence of locutions that is being processed, and the shared knowledge:

Definition 40 *An instantaneous description for a dialogue protocol is a tuple (q, α, SK) where $q \in Q$ is the current state, $\alpha \in (LS_{\mathcal{L}})^*$ is the remaining sequence of locutions, $SK \in \Gamma^*$ is the current shared knowledge.*

Definition 41
The relation $\vdash$ *satisfies* $(q, \rho_{id}(v^{(m)})\alpha, SK) \vdash (p, \alpha, newSK)$ *iff* $(p, newSK) \in \delta(q, \rho_{id}(v^{(m)}), SK)$.

We denote as $\Rightarrow^{+}_{W}$ and $\Rightarrow^{*}_{W}$ the transitive and reflexive closure of $\vdash$ in W.

Definition 42 *The language of dialogues generated by a system* $W \in ConvFST$ *specified as* $W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{L}}, \Gamma, \delta, q_0, SK_0, F)$ *is defined as:*

$$L_{Dg}(W) = \{\alpha \in (LS_{\mathcal{L}})^* \mid (q_0, \alpha, SK_0) \Rightarrow^{*}_{\Sigma} (q_f, \lambda, SK) \wedge q_f \in F\}.$$

Definition 43 *The language of share knowledge generated by a system* $W \in ConvFST$ *specified as* $W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{L}}, \Gamma, \delta, q_0, SK_0, F)$ *is defined as:*

$$L_{SK}(W) = \{ShareK \in \Gamma^* \mid (q_0, \alpha, SK_0) \Rightarrow^{*}_{\Sigma} (k, \beta, ShareK) \Rightarrow^{*}_{\Sigma} (q_f, \lambda, SK) \wedge q_f \in F\}.$$

If a dialogue protocol definition D can be defined by a $ConvFST$ system of degree n, $n \geq 1$, with language of share knowledge generated $L_{SK}(D)$ we write $D \in ConvFST_n$.

We denote by $ConvFST_n$, $n \geq 1$the family of languages $L(W)$ where $W \in ConvFST$ has degree n. When the number of agents is not bounded, then we replace the corresponding parameter with ∞.

When we are interested only in strings over some alphabet T, hence in the language $L_{SK}(W) \cap T^*$, then we speak about an *Extended Conversational Finite State Transition (EConvFST)* systems. We denote by $EConvFST_n$, $n \geq 1$ the family of languages $L_{SK}(W) \cap T^*$ where $W \in ConvFST_n$.

5.2.2 An example of dialogue protocol

This example of dialogue protocol specified by a $ConvFST$ system corresponds to an argumentation-based 2-party information seeking protocol, in the sense of [WK95], which we call $\mathcal{ISP}$. In such dialogue the Information-Seeker IS does not know the truth about a proposition p and asks an Information-Provider IP about p. If the IP knows if p is true or false, it will inform the IS and provide, upon request, the reasons that justify the value of p. The IS can accept or challenge the provided reasons. It can also happen that the IP does not know the value of truth of p, and in this case the dialogue finishes with the agent IS still not knowing the truth value of p.

Definition 44 *We define* $\mathcal{ISP}$ *as the tuple:*

$$\mathcal{ISP} = (K_{IS}, K_{IP}, \Sigma, Q, LS_{\mathcal{FOL}}, \Gamma, \delta, q_0, SK_0, F)$$

where:

- $\Sigma = PropVbles \cup Operators$, *where* $PropVbles$ *is a finite set of propositional variables and* $Operators = \{\rightarrow, \leftrightarrow, \vee, \wedge, \neg, (,), true, false\}$;
- $K_{IS} \in 2^{\Sigma^+}$ *and* $K_{IP} \in 2^{\Sigma^+}$ *are consistent sets of well defined propositional formulas over alphabet* Σ, *so it is not possible that* $(\exists \alpha \in K_i : K_i \models \alpha \wedge K_i \models \neg\alpha)$, $i \in \{IS, IP\}$. *Predicate* $\models : 2^{\Sigma^+} \times \Sigma^+ \rightarrow Boolean$ *is interpreted as logical deduction in propositional logic. Therefore* $K_i \models \varphi$ *indicates that the propositional formula* $\varphi \in \Sigma^+$ *can be deduced from the set of axioms* $K_i \in 2^{\Sigma^+}$ *using logical deduction in propositional logic;*

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$;
- $SK_0 = \lambda$;
- $\Gamma = \Sigma \cup \{?, !\}$;
- *We denote $\mathcal{FOL}$ the first order logic, then*
  $LS_{\mathcal{FOL}} = \left\{ \begin{array}{l} ask_i(\varphi), claim_i(\varphi), retract_i(), why_i(\varphi), argue_i(\varphi), concede_i(\varphi), \\ unknown_i(\varphi) \mid i \in \{IS, IP\} \wedge \varphi \in \Sigma^+ \wedge \varphi \text{ is a propositional formula} \end{array} \right\}$.

  Considering $i \in \{IS, IP\}$, SK the string of shared knowledge and variables φ, β, ψ propositional formulas over alphabet Σ, the semantic of the locutions in $LS_{\mathcal{FOL}}$ is the following:

$$\begin{array}{rcl}
prec(ask_i(\varphi)) & = & SK = \lambda \wedge (\exists\varphi : \neg(K_i \models \varphi) \wedge \neg(K_i \models \neg\varphi)) \\
prec(claim_i(\varphi)) & = & SK = \psi?\alpha \wedge (\exists\varphi : (K_i \models \varphi) \wedge (\varphi = \psi \vee \varphi = \neg\psi)) \\
prec(unknown_i(\varphi)) & = & SK = \varphi?\alpha \wedge \neg(K_i \models \varphi) \wedge \neg(K_i \models \neg\varphi) \\
prec(retract_i()) & = & SK = \varphi?\alpha \wedge \neg(\exists\beta : \neg(\beta \leftrightarrow \varphi) \wedge (K_i \models (\beta \rightarrow \varphi) \wedge \beta)) \\
prec(why_i(\alpha_2)) & = & SK = \psi\alpha \wedge \left[\begin{array}{l} \exists\alpha_1, \alpha_2 : (\psi \leftrightarrow \alpha_1 \wedge \alpha_2) \\ \wedge(K_i \models \alpha_1) \wedge \neg(K_i \models \alpha_2) \wedge \\ \neg \left[\begin{array}{l} \exists\alpha_3, \alpha_4 : (\alpha_2 \leftrightarrow \alpha_3 \wedge \alpha_4) \wedge (K_i \models \alpha_4) \wedge \\ (\alpha_4 \leftrightarrow false) \end{array} \right] \end{array} \right] \\
prec(argue_i(\varphi)) & = & SK = \psi?\alpha \wedge (\exists\varphi : (K_i \models (\varphi \rightarrow \psi) \wedge \varphi)) \\
prec(concede_i(\varphi)) & = & SK = \varphi\alpha \wedge (\exists\varphi : (K_i \models \varphi))
\end{array}$$

- *Considering variables φ, ψ as propositional formulas over alphabet Σ, and shared knowledge $T \in \Gamma^*$, then function δ is defined in the following way:*
  $\delta(q_0, ask_{IS}(\varphi), SK_0) = \{(q_1, \varphi?)\}$
  $\delta(q_1, claim_{IP}(\varphi), \psi?\, T) = \{(q_2, \varphi\, \psi?\, T)\}$
  $\delta(q_1, unknown_{IP}(\psi), \psi?\, T) = \{(q_5, \psi?\, T)\}$
  $\delta(q_2, why_{IS}(\varphi), \psi\, T) = \{(q_4, \varphi?\, \psi\, T)\}$
  $\delta(q_4, argue_{IP}(\varphi), \psi?\, T) = \{(q_2, \varphi\, \psi?\, T)\}$
  $\delta(q_2, concede_{IS}(\varphi), \varphi\, T) = \{(q_3, \varphi!\, \varphi\, T)\}$
  $\delta(q_4, retract_{IP}(), \psi?\, T) = \{(q_5, \psi?\, T)\}$, *and*
- $F = \{q_3, q_5\}$.

The interchange of locutions in $\mathcal{ISP}$ can be represented by the graph in figure 5.1.

From the definition of $\mathcal{ISP}$, it immediately follows that $\mathcal{ISP} \in ConvFST_2$.

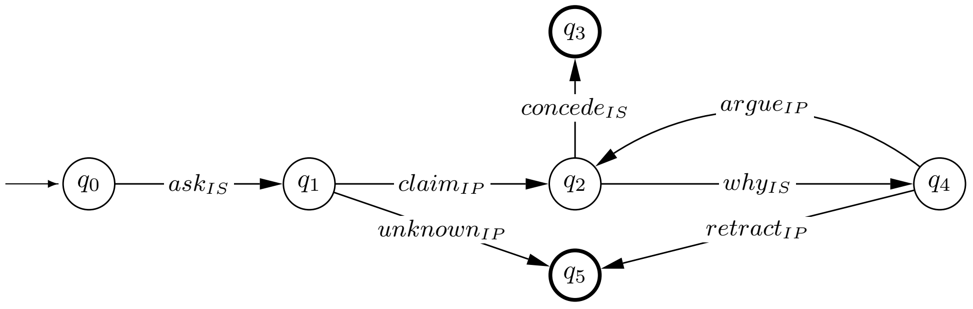

Figure 5.1: Transition graph for the interchange of locutions in $\mathcal{ISP}$

Once a dialogue protocol is formally specified in a formal framework, in this case the a *ConvFST* system, some formal analysis and proofs can be done. For dialogue protocol $\mathcal{ISP}$ we prove which is the set of dialogue instances that it defines:

Theorem 14 *The set of dialogue instances satisfying $\mathcal{ISP} \in ConvFST_2$ protocol is given by*

$$
\begin{array}{rl}
L_{Dg}(\mathcal{ISP}) = & \left\{ \begin{array}{l} ask_{IS}(\varphi)unknown_{IP}(\varphi) \mid \neg(K_{IS} \models \varphi) \wedge \neg(K_{IS} \models \neg\varphi) \\ \wedge\neg(K_{IP} \models \varphi) \wedge \neg(K_{IP} \models \neg\varphi) \end{array} \right\} \cup \\
& \left\{ \begin{array}{l} ask_{IS}(\varphi)claim_{IP}(\alpha_1)why_{IS}(\beta_1)argue_{IP}(\alpha_2)why_{IS,2}(\beta_2)... \\ why_{IS}(\beta_n)argue_{IP}(\alpha_{n+1})concede_{IS}(\alpha_{n+1}) \mid \neg(K_{IS} \models \varphi)\wedge \\ \neg(K_{IS} \models \neg\varphi) \wedge (\alpha_1 = \varphi \vee \alpha_1 = \neg\varphi) \wedge (K_{IP} \models \alpha_1)\wedge \\ \left(\forall 1 \le i \le n : \left(\begin{array}{l} \exists\beta_i, \psi_i : (\alpha_i \leftrightarrow \beta_i \wedge \psi_i)\wedge \\ \neg(K_{IS} \models \beta_i) \wedge (K_{IS} \models \psi_i)\wedge \\ (K_{IP} \models (\alpha_{i+1} \rightarrow \beta_i) \wedge \alpha_{i+1})\wedge \\ \neg\left(\begin{array}{l} \exists\pi_i, \gamma_i : (\beta_i \leftrightarrow \pi_i \wedge \gamma_i)\wedge \\ (K_{IS} \models \gamma_i) \wedge (\gamma_i \leftrightarrow false) \end{array} \right) \end{array} \right) \right) \\ \wedge(K_{IS} \models \alpha_{n+1}) \end{array} \right\} \cup \\
& \left\{ \begin{array}{l} ask_{IS}(\varphi)claim_{IP}(\alpha_1)why_{IS}(\beta_1)argue_{IP}(\alpha_2)why_{IS}(\beta_2)... \\ argue_{IP}(\alpha_{n+1})why_{IS}(\beta_{n+1})retract_{IP}() \mid \neg(K_{IS} \models \varphi)\wedge \\ \neg(K_{IS} \models \neg\varphi) \wedge (\alpha_1 = \varphi \vee \alpha_1 = \neg\varphi) \wedge (K_{IP} \models \alpha_1)\wedge \\ \left(\forall 1 \le i \le n+1 : \left(\begin{array}{l} \exists\beta_i, \psi_i : (\alpha_i \leftrightarrow \beta_i \wedge \psi_i) \wedge \neg(K_{IS} \models \beta_i)\wedge \\ (K_{IS} \models \psi_i)\wedge \\ \neg\left(\begin{array}{l} \exists\pi_i, \gamma_i : (\beta_i \leftrightarrow \pi_i \wedge \gamma_i)\wedge \\ (K_{IS} \models \gamma_i) \wedge \neg(\gamma_i \leftrightarrow true) \end{array} \right) \\ \left(i < n+1 \rightarrow (K_{IP} \models (\alpha_{i+1} \rightarrow \beta_i) \wedge \alpha_{i+1}) \right) \end{array} \right) \right) \wedge \\ \left(i = n+1 \rightarrow \neg\left(\begin{array}{l} \exists\alpha_{n+2} : \neg(\alpha_{n+2} \leftrightarrow \beta_{n+1})\wedge \\ (K_{IP} \models (\alpha_{n+2} \rightarrow \beta_{n+1}) \wedge \alpha_{n+2}) \end{array} \right) \right) \end{array} \right\}.
\end{array}
$$

Proof.

We prove first the right inclusion. Every dialogue instance $\beta \in L_{Dg}(\mathcal{ISP})$ satisfies that $\beta = ask_{IS}(\varphi)\alpha$ for some $\varphi \in \Sigma^+$ where $\neg(K_{IS} \models \varphi) \wedge \neg(K_{IS} \models \neg\varphi)$. The following is a tautology in propositional logic:

$$[\neg(K_{IP} \models \varphi) \wedge \neg(K_{IP} \models \neg\varphi)] \vee [(K_{IP} \models \varphi) \vee (K_{IP} \models \neg\varphi)] \ .$$

Then the IP can only answer in one of these ways:

1. If $\neg(K_{IP} \models \varphi) \wedge \neg(K_{IP} \models \neg\varphi)$ then the IP answers back $unkown_{IP}()$. In this case we obtain a string from the first subset of $L_{Dg}(\mathcal{ISP})$.

2. If $(\alpha_1 = \varphi \vee \alpha_1 = \neg\varphi) \wedge (K_{IP} \models \alpha_1)$ then the IP answers back $claim_{IP}(\alpha_1)$. The following is a tautology: $(K_{IS} \models \alpha_1) \vee \neg(K_{IS} \models \alpha_1)$. Then the IS can only answer back in one of these ways:

   (a) If $(K_{IS} \models \alpha_1)$ then the IS answers back $concede_{IS}(\alpha_1)$. In this case we obtain a string from the second subset of $L_{Dg}(\mathcal{ISP})$.

   (b) If $\neg(K_{IS} \models \alpha_1)$ then this is also a tautology:
   $$\neg(K_{IS} \models \alpha_1) \rightarrow \left(\begin{array}{l} \exists\beta_1, \psi_1 : (\alpha_1 \leftrightarrow \beta_1 \wedge \psi_1) \wedge \neg(K_{IS} \models \beta_1) \wedge (K_{IS} \models \psi_1) \wedge \\ \neg\Big(\exists\pi_1, \gamma_1 : (\beta_1 \leftrightarrow \pi_1 \wedge \gamma_1) \wedge (K_{IS} \models \gamma_1) \wedge (\gamma_1 \leftrightarrow false) \Big) \end{array} \right).$$
   Then the IS answers back $why_{IS}(\beta_1)$.
   The following is a tautology:
   $$\begin{array}{l} \neg\Big(\exists\alpha_2 : \neg(\alpha_2 \leftrightarrow \beta_1) \wedge (K_{IP} \models (\alpha_2 \rightarrow \beta_1) \wedge \alpha_2) \Big) \vee \\ \Big(\exists\alpha_2 : \neg(\alpha_2 \leftrightarrow \beta_1) \wedge (K_{IP} \models (\alpha_2 \rightarrow \beta_1) \wedge \alpha_2) \Big) \end{array}$$
   Then the IP can only answer back in one of these ways:

      i. If $\neg\Big(\exists\alpha_2 : \neg(\alpha_2 \leftrightarrow \beta_1) \wedge K_{IP} \models (\alpha_2 \rightarrow \beta_1) \wedge \alpha_2 \Big)$ then the IP answers back $retract_{IP}()$. In this case we obtain a string from the third subset of $L_{Dg}(\mathcal{ISP})$.

      ii. If $\Big(\exists\alpha_2 : \neg(\alpha_2 \leftrightarrow \beta_1) \wedge (K_{IP} \models (\alpha_2 \rightarrow \beta_1) \wedge \alpha_2) \Big)$ then the IP answers back $argue_{IP}(\alpha_2)$.
      The following is a tautology: $(K_{IS} \models \alpha_2) \vee \neg(K_{IS} \models \alpha_2)$. Then the IS can only answer back as described in 2.a or in 2.b, where propositions $\alpha_1, \beta_1, \psi_1, \pi_1, \gamma_1$ are respectively replaced for propositions $\alpha_2, \beta_2, \psi_2, \pi_2, \gamma_2$.

We now prove the left inclusion.

1. If $\alpha \in \left\{ \begin{array}{l} ask_{IS}(\varphi)unknown_{IP}(\varphi) \mid \neg(K_{IS} \models \varphi) \wedge \neg(K_{IS} \models \neg\varphi) \\ \wedge\neg(K_{IP} \models \varphi) \wedge \neg(K_{IP} \models \neg\varphi) \end{array} \right\}$

   then $\alpha \in L_{Dg}(\mathcal{ISP})$ because $(q_0, \alpha, SK_0) \Rightarrow^2_{\mathcal{ISP}} (q_5, \lambda, SK_f)$.

2. If $\alpha \in \left\{ \begin{array}{l} ask_{IS}(\varphi)claim_{IP}(\alpha_1)why_{IS}(\beta_1)argue_{IP}(\alpha_2)why_{IS,2}(\beta_2)... \\ why_{IS}(\beta_n)argue_{IP}(\alpha_{n+1})concede_{IS}(\alpha_{n+1}) \mid \neg(K_{IS} \models \varphi) \wedge \\ \neg(K_{IS} \models \neg\varphi) \wedge (\alpha_1 = \varphi \vee \alpha_1 = \neg\varphi) \wedge (K_{IP} \models \alpha_1) \wedge \\ \left(\forall 1 \leq i \leq n : \left(\begin{array}{l} \exists\beta_i, \psi_i : (\alpha_i \leftrightarrow \beta_i \wedge \psi_i) \wedge \\ \neg(K_{IS} \models \beta_i) \wedge (K_{IS} \models \psi_i) \wedge \\ (K_{IP} \models (\alpha_{i+1} \rightarrow \beta_i) \wedge \alpha_{i+1}) \wedge \\ \neg\left(\begin{array}{l} \exists\pi_i, \gamma_i : (\beta_i \leftrightarrow \pi_i \wedge \gamma_i) \wedge \\ (K_{IS} \models \gamma_i) \wedge (\gamma_i \leftrightarrow false) \end{array} \right) \end{array} \right) \right) \\ \wedge(K_{IS} \models \alpha_{n+1}) \end{array} \right\}$

then $\alpha \in L_{Dg}(\mathcal{ISP})$ because $(q_0, \alpha, SK_0) \Rightarrow^{+}_{\mathcal{ISP}} (q_3, \lambda, SK_f)$.

3. If
$$\alpha \in \left\{ \begin{array}{l} ask_{IS}(\varphi)claim_{IP}(\alpha_1)why_{IS}(\beta_1)argue_{IP}(\alpha_2)why_{IS}(\beta_2)... \\ argue_{IP}(\alpha_{n+1})why_{IS}(\beta_{n+1})retract_{IP}() \mid \neg(K_{IS} \models \varphi) \wedge \\ \neg(K_{IS} \models \neg\varphi) \wedge (\alpha_1 = \varphi \vee \alpha_1 = \neg\varphi) \wedge (K_{IP} \models \alpha_1) \wedge \\ \left(\forall 1 \leq i \leq n+1 : \left(\begin{array}{l} \exists \beta_i, \psi_i : (\alpha_i \leftrightarrow \beta_i \wedge \psi_i) \wedge \neg(K_{IS} \models \beta_i) \wedge \\ (K_{IS} \models \psi_i) \wedge \\ \neg \left(\begin{array}{l} \exists \pi_i, \gamma_i : (\beta_i \leftrightarrow \pi_i \wedge \gamma_i) \wedge \\ (K_{IS} \models \gamma_i) \wedge \neg(\gamma_i \leftrightarrow true) \end{array} \right) \\ \left(i < n+1 \rightarrow (K_{IP} \models (\alpha_{i+1} \rightarrow \beta_i) \wedge \alpha_{i+1}) \right) \end{array} \right) \right) \wedge \\ \left(i = n+1 \rightarrow \neg \left(\begin{array}{l} \exists \alpha_{n+2} : \neg(\alpha_{n+2} \leftrightarrow \beta_{n+1}) \wedge \\ (K_{IP} \models (\alpha_{n+2} \rightarrow \beta_{n+1}) \wedge \alpha_{n+2}) \end{array} \right) \right) \end{array} \right\}$$

then $\alpha \in L_{Dg}(\mathcal{ISP})$ because $(q_0, \alpha, SK_0) \Rightarrow^{+}_{\mathcal{ISP}} (q_5, \lambda, SK_f)$.

■

Once we have the formal definition of the language of dialogues generated by the protocol $\mathcal{ISP}$ we prove that if speakers IS and IP have no knowledge in common, then the IS will never get a satisfactory answer to his question:

Corollary 5 *The protocol $\mathcal{ISP} \in ConvFST_2$ satisfies:*

$$\left[\begin{array}{l} \neg(K_{IS} \models \varphi) \wedge \neg(K_{IS} \models \neg\varphi) \wedge (K_{IP} \models \psi) \\ \wedge (\psi = \varphi \vee \psi = \neg\varphi) \wedge \neg(\exists \alpha_k : (K_{IS} \models \alpha_k) \wedge (K_{IP} \models \alpha_k)) \end{array} \right] \rightarrow$$

$$L_{Dg}(\mathcal{ISP}) = \left\{ \begin{array}{l} ask_{IS}(\varphi)unknown_{IP}(\varphi) \mid \neg(K_{IS} \models \varphi) \wedge \neg(K_{IS} \models \neg\varphi) \\ \wedge\neg(K_{IP} \models \varphi) \wedge \neg(K_{IP} \models \neg\varphi) \end{array} \right\} \cup$$
$$\left\{ \begin{array}{l} ask_{IS}(\varphi)claim_{IP}(\alpha_1)why_{IS}(\beta_1)argue_{IP}(\alpha_2)why_{IS}(\beta_2)... \\ argue_{IP}(\alpha_{n+1})why_{IS}(\beta_{n+1})retract_{IP}() \mid \neg(K_{IS} \models \varphi) \wedge \\ \neg(K_{IS} \models \neg\varphi) \wedge (\alpha_1 = \varphi \vee \alpha_1 = \neg\varphi) \wedge (K_{IP} \models \alpha_1) \wedge \\ \left(\forall 1 \leq i \leq n+1 : \left(\begin{array}{l} \exists \beta_i, \psi_i : (\alpha_i \leftrightarrow \beta_i \wedge \psi_i) \wedge \neg(K_{IS} \models \beta_i) \wedge \\ (K_{IS} \models \psi_i) \wedge \\ \neg \left(\begin{array}{l} \exists \pi_i, \gamma_i : (\beta_i \leftrightarrow \pi_i \wedge \gamma_i) \wedge \\ (K_{IS} \models \gamma_i) \wedge (\gamma_i \leftrightarrow false) \end{array} \right) \\ \left(i < n+1 \rightarrow (K_{IP} \models (\alpha_{i+1} \rightarrow \beta_i) \wedge \alpha_{i+1}) \right) \end{array} \right) \right) \wedge \\ \left(i = n+1 \rightarrow \neg \left(\begin{array}{l} \exists \alpha_{n+2} : \neg(\alpha_{n+2} \leftrightarrow \beta_{n+1}) \wedge \\ (K_{IP} \models (\alpha_{n+2} \rightarrow \beta_{n+1}) \wedge \alpha_{n+2}) \end{array} \right) \right) \end{array} \right\}$$

Proof. We prove this by reductio ad absurdum. Let us suppose that the dialogue instance below satisfies $\mathcal{ISP}$.

$$ask_{IS}(\varphi)claim_{IP}(\alpha_1)why_{IS,1}(\beta_1)argue_{IP,1}(\alpha_2)why_{IS,2}(\beta_2)\dots$$
$$why_{IS,n}(\beta_n)argue_{IP,n}(\alpha_{n+1})concede_{IS}(\alpha_{n+1})$$

From theorem 14 this means that $(K_{IP} \models (\alpha_{n+1} \rightarrow \beta_n) \wedge \alpha_{n+1}) \wedge (K_{IS} \models \alpha_{n+1})$. In particular $[(K_{IS} \models \alpha_{n+1}) \wedge (K_{IP} \models \alpha_{n+1})]$ contradicts our hypothesis, and proves the result. ■

The converse of this result does not hold. For example take the the set of axioms $K_{IP} = \{k, p, s \rightarrow p\}$, $K_{IS} = \{k\}$ and the dialogue instance

$$[d = ask_{IS}(p)claim_{IP}(p)why_{IS}(p)argue_{IP}(s \rightarrow p)why_{IS}(s)retract_{IP}()]$$

5.2.3 A possible application in Argumentation-based Dialogue Theory

In the 80s different *logics for defeasible argumentation* were developed to formalize the way humans reason deriving tentative conclusions on the basis of uncertain or incomplete information, conclusions that can be withdrawn when some more information is available. Examples of defeasible or non monotonic logics are [Pol74], [Dun95], [PV00]. Based on these logics, *logical argumentation systems* were introduced to formalize this kind of reasoning in terms of the interactions between arguments for alternative conclusions. Non-monotonicity arises since arguments can be defeated by stronger counterarguments.

Artificial Intelligence has studied the exchange of *arguments and counterarguments in the context of multi-agent interaction*. Few approaches have focused on the meta-theoretical study of dialogue protocols for multi-agent systems. A general and abstract formal framework for the definition of argumentation dialogue protocols where no assumption is made about the locutions uttered, the commitments taken by agents and the argumentation system involved is presented in [ABP06]. Although there are some other proposals of argumentation-based formal frameworks for multi-agent systems, they take place in particular settings where some parameters are fixed, like: the locutions uttered, the commitments taken by the agents during the dialogues or the argumentation systems that are involved. For instance Reed [Ree98], Gordon [Gor94, GK97], Kraus [KNS93],[ABP06] and Parsons [PWA03b].

The work from [PWA03b] corresponds to a logic-based formalism for modeling dialogues between intelligent and autonomous software agents, built on a theory of abstract dialogue games. The underlying argumentation system in this framework was introduced by Amgoud in [Amg98]. In [PWA03b] [PWA02] they present formal results studying the consequences that the adoption of the argumentation system from [Amg98] can have over the definition of protocols in their framework. For instance they studied time complexity and the influence that the combination of certain types of agents can have over the execution and termination of a protocol.

The definition and formal study of argumentation-based frameworks for multi-agent systems is based on the necessity of providing generic, abstract and well defined formalisms in which protocols can be specified, compared and evaluated in a common notation in a precise way. These studies are fundamental if we want to address open problems of the type introduced in [PWA03b]:

- How might one choose between two protocols?;
- When is one protocol preferable to another?;
- When do two protocols differ?;
- Can we tell if a protocol is new (in the sense of providing a different functionality from an existing protocol rather than just having equivalent locution with different names)?;

- Is a protocol new (in the sense of providing a different functionality from an existing protocol rather than just having equivalent locutions with different names)?.

Below we briefly introduce the formal system of argumentation due to Amgoud [Amg98] that forms the backbone of the framework introduced in [PWA03b]. The work of Amgoud is inspired by the work of Dung [Dun95] but goes further in dealing with preferences between arguments.

Amgoud's argumentation formal system

A possibly inconsistent knowledge base Σ with no deductive closure is considered. It is assumed that Σ contains formulas in $\mathcal{PL_U}$. $\mathcal{PL_U}$ denotes the propositional language where apart from constants true and false the constant $\mathcal{U}$ is considered to denote uncertainty. $\vdash$ stands for classical inference, $\rightarrow$ for logical implication, and $\leftrightarrow$ for logical equivalence. An argument is defined as:

Definition 45 *An* argument *is a pair $A = (H, h)$ where h is a formula of $\mathcal{PL_U}$ and H a subset of Σ such that:*

1. *H is consistent;*
2. *$H \vdash h$; and*
3. *H is minimal, so no proper subset of H satisfying both 1 and 2 exists.*

H is called the support *of A, written H = Support(A) and h is the* conclusion *of A written h = Conclusion(A).*

We talk of h being *supported* by the argument (H, h)

In general, since Σ is inconsistent, arguments in $\mathcal{A}(\Sigma)$, the set of all arguments which can be made from Σ, will conflict, therefore the notion of undercutting:

Definition 46 *Let A_1 and A_2 be two arguments of $\mathcal{A}(\Sigma)$. A_1* undercuts *A_2 iff $\exists h \in Support(A_2)$ such that $h \leftrightarrow \neg Conclusion(A_1)$.*

In other words, an argument is undercut if and only if there is another argument which has as its conclusion the negation of an element of the support for the first argument.

To capture the fact that some facts are more strongly believed it is assumed that any set of facts has a preference order over it. It is supposed that this ordering derives from the fact that the knowledge base Σ is stratified into non-overlapping sets $\Sigma_1, \ldots, \Sigma_n$ such that facts in Σ_i are all equally preferred and are more preferred than those in Σ_j where $j > i$. The preference level of a nonempty subset H of Σ, $level(H)$, is the number of the highest numbered layer which has a member in H.

Definition 47 *Let A_1 and A_2 be two arguments in $\mathcal{A}(\Sigma)$. A_1 is* preferred *to A_2 according to Pref, $Pref(A_1, A_2)$, iff $level(Support(A_1)) \leq level(Support(A_2))$.*

$\gg^{Pref}$ denotes the strict pre-order associated with *Pref*. If A_1 is preferred to A_2, it is said that A_1 is *stronger* than A_2. An argumentation system is defined in the following way:

Definition 48 *An* argumentation system *is a triple* $(\mathcal{A}(\Sigma), Undercut, Pref)$ *such that:*

- $\mathcal{A}(\Sigma)$ *is a set of the arguments built from* Σ*,*
- *Undercut is a binary relation representing the defeat relationship between arguments,* $Undercut \subseteq \mathcal{A}(\Sigma) \times \mathcal{A}(\Sigma)$*, and*
- *Pref is a (partial or complete) preordering on* $\mathcal{A}(\Sigma) \times \mathcal{A}(\Sigma)$*.*

The preference order makes it possible to distinguish different types of relation between arguments:

Definition 49 *Let* A_1*,* A_2 *be two arguments of* $\mathcal{A}(\Sigma)$*.*

- *If* A_2 *undercuts* A_1 *then* A_1 defends itself *against* A_2 *iff* $A_1 \gg^{Pref} A_2$*. Otherwise,* A_1 does not defend itself.
- *A set of arguments* $\mathcal{S}$ defends *A iff:* $\forall$ *B undercuts A and A does not defend itself against B then* $\exists$ $C \in \mathcal{S}$ *such that C undercuts B and B does not defend itself against C.*

Henceforth, $C_{Undercut,Pref}$ will gather all non-undercut arguments and arguments defending themselves against all their undercutting arguments. In [Amg98], Amgoud showed that the set $\underline{\mathcal{S}}$ of acceptable arguments of the argumentation system $\langle \mathcal{A}(\Sigma), Undercut, Pref \rangle$ is the least fixpoint of a function $\mathcal{F}$:

$$\begin{array}{rcl} \mathcal{S} & \subseteq & \mathcal{A}(\Sigma) \\ \mathcal{F}(\mathcal{S}) & = & \{(H,h) \in \mathcal{A}(\Sigma) | (H,h) \text{ is defended by } \mathcal{S}\} \end{array}$$

Definition 50 *The set of* acceptable *arguments for an argumentation system* $\langle \mathcal{A}(\Sigma), Undercut, Pref \rangle$ *is:*

$$\begin{array}{rcl} \underline{\mathcal{S}} & = & \bigcup \mathcal{F}_{i \geq 0}(\emptyset) \\ \underline{\mathcal{S}} & = & C_{Undercut,Pref} \cup \left[\bigcup \mathcal{F}_{i \geq 1}(C_{Undercut,Pref}) \right] \end{array}$$

An argument is acceptable *if it is a member of the acceptable set.*

An acceptable argument is one which is, in some sense, proven since all the arguments which might undermine it are themselves undermined. However, this status can be revoked following the discovery of a new argument (possibly as the result of the communication of some new information from other agent).

The set of locutions considered in [PWA03b] is the following:

$$DgArg_{\mathcal{FOL}} = \left\{ \begin{array}{l} assert(p), assert(S), accept(p), challenge(p), question(p), reject(p) \mid \\ p \text{ is a formula in } \mathcal{PL_U} \wedge S \text{ is a set of formulas in } \mathcal{PL_U} \end{array} \right\}$$

The semantic of each locution from $DgArg_{\mathcal{PL}}$ is given by the effect that it has on the commitment store of the agent that makes the locution (the other agent's commitment store is unchanged). Each locution in $DgArg_{\mathcal{PL}}$ is defined as follows:

assert(p)

$$CS_i(P) = CS_{i-1}(P) \cup \{p\}$$

where p can be any propositional formula, as well as the special character $\mathcal{U}$ which we have already met.

assert(S)

$$CS_i(P) = CS(P)_{i-1} \cup S$$

where S is a set of formulas representing the support of an argument.

accept(p)

$$CS_i(P) = CS_{i-1}(P) \cup \{p\}$$

where p is a propositional formula.

challenge(p)

$$CS_i(P) = CS_{i-1}(P)$$

where p is a propositional formula.

question(p)

$$CS_i(P) = CS_{i-1}(P)$$

where p is a propositional formula.

Finally, the locution $reject(p)$ does not change the commitment stores, but indicates that the agent that utters it will not $accept\ p$.

Which locutions can be made at a particular time depends on the protocol and also *assertion* and *acceptance* attitudes.

Definition 51 *An agent may have one of three* assertion *attitudes.*

- *a* confident *agent can assert any proposition p for which it can construct an argument* (S, p).
- *a* careful *agent can assert any proposition p for which it can construct an argument, if it is unable to construct a stronger argument for* $\neg p$.
- *a* thoughtful *agent can assert any proposition p for which it can construct an acceptable argument* (S, p).

Definition 52 *An agent may have one of three* acceptance *attitudes.*

- *a* credulous *agent can accept any proposition p if it is backed by an argument.*

- *a* cautious *agent can accept any proposition p that is backed by an argument if it is unable to construct a stronger argument for* $\neg p$.
- *a* skeptical *agent can accept any proposition p if it is backed by an acceptable argument.*

Since agents are typically involved in both asserting and accepting propositions, they denote the combination of an agent's two attitudes as

$$\langle \textit{assertion attitude} \rangle / \langle \textit{acceptance attitude} \rangle$$

An example of an argumentation-based dialogue protocol

Definition 53 *We call $\mathcal{I}_{CS}$ the following inquiry dialogue protocol defined in the Amgoud's argumentation formal system, that resembles the inquiry dialogue protocol from [PMW04]. According to [WK95] in an inquiry dialogue protocol the participants collaborate to answer some question whose answer they do not know. In $\mathcal{I}_{CS}$ we assume that two agents A and B have already agreed to engage in an inquiry about some proposition p, maybe by some other dialogue protocol, and from this point they can adopt the following protocol:*

1. *A asserts $q \rightarrow p$ for some q, or A asserts $\mathcal{U}$ and the dialogue terminates.*
2. *B accepts $q \rightarrow p$ if its acceptance attitude allows, or challenges it.*
3. *A replies to a challenge with an assert(S), where S is the support of an argument for the last proposition challenged by B.*
4. *Following a challenge, goto 2 for each proposition $s \in S$ in turn, replacing $q \rightarrow p$ by s.*
5. *If there are no more $s \in S$ to be challenged by B then continue to 6.*
6. *If $\mathcal{A}(CS(A) \cup CS(B))$ includes an argument for $q \rightarrow p$ which is acceptable to both agents, then first A and then B accept p, and the dialogue terminates successfully. Else continue to 7.*
7. *Go to 1, reversing the roles of A and B and substituting $q \rightarrow p$ for $r \rightarrow q$ for some r.*

Here the initial conditions of the dialogue are that neither agent has an argument for p.

We prove that the inquiry dialogue $\mathcal{I}_{CS}$ can be specified by a $ConvFST_2$ system, considering the following definition of simulation:

Definition 54 *An argumentation-based dialogue protocol definition P expressed in the framework from [PWA03b] can be specified by a ConvFST system iff*

$$(\exists M \in ConvFST : (\forall \alpha \in (DgArg_{\mathcal{FOL}})^* : \alpha \textit{ satisfies protocol } P \leftrightarrow \alpha \in L_{Dg}(M))).$$

The dialogue protocol $\mathcal{I}_{CS}$ from definition 53 can be specified using the framework $ConvFST_2$. We propose to simulate protocol $\mathcal{I}_{CS}$ with the dialogue protocol $IQ \in ConvFST_2$, where:

$$IQ = (K_A, K_B, \Upsilon, Q, LS_{\mathcal{FOL}}, \Gamma, \delta_{IQ}, q_0, SK_0, F)$$

with:

- $\Upsilon = PropVbles \cup \{\vee, \wedge, \leftrightarrow, \rightarrow, \mathcal{U}, Pref, undercuts, (,)\}$ where $PropVbles$ is a finite set of propositional variables.
- $K_A = \{\alpha_{\Sigma_A}, \beta_{Undercut_A}, \gamma_{Pref_A}\}$ where:
  - $\langle \Sigma_A, Undercut_A, Pref_A \rangle$ is an argumentation system as in definition 48,
  - α_{Σ_A} is the string representation of Σ_A,
  - $\beta_{Undercut_A}$ is the string representation of $Undercut_A$ and
  - γ_{Pref_A} is the string representation of $Pref_A$.

  K_B is similarly defined.
- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}\}$.
- $LS_{\mathcal{FOL}} = \left\{ \begin{array}{l} \lambda_i, assert_i(p),\ assert_i(S),\ accept_i(p),\ challenge_i(p)\ | \\ p \text{ is a formula in } \mathcal{PL_U} \wedge \\ S \text{ is a set of formulas in } \mathcal{PL_U} \wedge i \in \{A, B\} \end{array} \right\}$

  where $\mathcal{PL_U}$ denotes the propositional logic with constants true, false and $\mathcal{U}$; the locutions λ_j, $j \in \{A, B\}$, are identity elements of $LS_{\mathcal{FOL}}$:
  $(\forall \lambda_j, \rho_i(\phi^{(h)}) \in LS_{\mathcal{FOL}} : \lambda_j \rho_i(\phi^{(h)}) = \rho_i(\phi^{(h)})\lambda_j = \rho_i(\phi^{(h)}))$.

  The preconditions associated with the locutions in $LS_{\mathcal{FOL}}$ are the result of the formalization of the preconditions of the locutions in $\mathcal{I}_{CS}$ plus the definition of a precondition for $\lambda_i \in LS_{\mathcal{FOL}}$. The precondition for $\lambda_i \in LS_{\mathcal{FOL}}$ corresponds to the condition that is checked in the step 6 of protocol $\mathcal{I}_{CS}$. While in $\mathcal{I}_{CS}$ the preconditions of the locutions are expressed in natural language, in $LS_{\mathcal{FOL}}$ they are expressed in first order logic. String $SK \in \Gamma^*$ corresponds to the shared knowledge and $SK = \alpha\{\beta\}_A\{\eta\}_B$. In SK the string α corresponds to uttered locutions, such that the leftmost substring in α corresponds to the last uttered locution. In SK the strings β and η correspond to the string representation of $CS(A)$ and $CS(B)$, the commitments acquired by agents A and B respectively. The preconditions for the locutions in $LS_{\mathcal{FOL}}$, considering $i, j \in \{A, B\}$, $i \neq j$, and predicates Ass_i and Acc_i introduced in definitions 51 and 52 over argumentation system $\langle \mathcal{A}(\Sigma_i \cup CS(j)), Undercut_i, Pref_i \rangle$ are:

$$\begin{array}{lcl}
prec(accert_i(\mathcal{U})) & = & [SK = \langle initial, p\rangle\alpha \vee SK = \langle challenge, p\rangle_i\alpha] \wedge \\
 & & \neg[\exists (T, q \rightarrow p) \in \mathcal{A}(\Sigma_i) : Ass_i(T, q \rightarrow p)]] \\
prec(assert_i(q \rightarrow p)) & = & (SK = \langle initial, p\rangle\alpha \vee SK = \langle contribution, p \rightarrow r\rangle_j\alpha) \wedge \\
 & & [\exists (T, q \rightarrow p) \in \mathcal{A}(\Sigma_i) : Ass_i(T, q \rightarrow p)] \\
prec(assert_i(T)) & = & SK = \langle challenge, p\rangle_j\alpha \wedge [\exists (T, p) \in \mathcal{A}(\Sigma_i) : Ass_i(T, p)] \\
prec(accept_i(p)) & = & (SK = \langle assert, p\rangle_j\alpha \wedge Acc_i(p) \\
prec(accept_i(p)) & = & (SK = \langle contribution, p\rangle_i\alpha \vee SK = \langle termination, p\rangle_j\alpha) \\
 & & \wedge Acc_i(p) \\
prec(challenge_i(p)) & = & SK = \langle assert, p\rangle_j\alpha \wedge \neg(Acc_j(p)) \\
prec(\lambda_i) & = & SK = \langle termination, q \rightarrow p\rangle_j\alpha \wedge \neg(Acc_i(q \rightarrow p))
\end{array}$$

- $\Gamma = \{\{_A, \{_B, \}_A, \}_B\} \cup \left\{ \begin{array}{l} \langle initial, p\rangle, \langle \rho, p\rangle_i, \langle termination, p \rightarrow q\rangle_i, \langle contribution, p \rightarrow q\rangle_i \mid \\ p, q \text{ are formulas in } \mathcal{PL_U} \wedge \rho_i(p) \in LS_{\mathcal{FOL}} \wedge i \in \{A, B\} \end{array} \right\}$

  When an agent with identifier $i \in \{A, B\}$ asserts a proposition $p \rightarrow q$ that contributes to the proof under construction, then the leftmost substring in SK representing a locution is replaced by $\langle assert, p \rightarrow q\rangle_i$ followed by $\langle contribution, p \rightarrow q\rangle_i$. The second string is added to SK not to lose the last contribution $p \rightarrow q$ to the common proof once the first string disappears from SK when the agent with identifier $j \in \{A, B\}$ accepts $p \rightarrow q$ or it accepts the propositions that justify $p \rightarrow q$. While if the agent with identifier i asserts a proposition q that does not contribute to the common proof but that is provided as justification of a challenged proposition, then only the string $\langle assert, q\rangle_i$ substitutes the leftmost substring in SK that represents the last uttered locution. When the string $\langle contribution, p \rightarrow q\rangle_i$ is the leftmost substring in SK, it means that the agent with identifier j has accepted the proposition $p \rightarrow q$ asserted by the agent i or that it has accepted all the justifications that agent i provided for $p \rightarrow q$. When the string $\langle termination, p \rightarrow q\rangle_i$ is the leftmost substring in SK, it means two things. On one hand it means that the agent j has accepted the proposition $p \rightarrow q$ asserted by agent i or that it has accepted all the justifications that agent i provided for $p \rightarrow q$. On the other hand it means that the agent i has also found in $CS(i) \cup CS(j)$ the justification for $p \rightarrow q$ and it has accepted it.

- $SK_0 = \langle initial, p\rangle\{\}_A\{\}_B$

- For each dialogue move in $\mathcal{I}_{CS}$ if an agent with identifier $i \in \{A, B\}$ utters a locution $\rho(\phi_1, ..., \phi_n)$ a transition labeled with locution $\rho_i(\phi_1, ..., \phi_n)$ is defined in δ_{IQ}. Besides each transition that is triggered in IQ produces some effect over string SK. For all the locutions from $LS_{\mathcal{FOL}}$, expect the ones used for assertion and acceptance of a proposition, there is no effect over the substrings from SK that represent commitment stores $CS(A)$ and $CS(B)$. When an agent $i \in \{A, B\}$ asserts a proposition in any state of IQ or accepts a proposition in the transition from state q_2 to q_2 or from state q_7 to q_7 (step 2 of $\mathcal{I}_{CS}$), there is an effect over SK. The effect is the addition of the string representation of the asserted proposition in the string representation of set $CS(i)$, to denote that the proposition is inserted into the commitment store of agent i. When an agent $i \in \{A, B\}$ accepts a proposition in the transition from state q_2 to q_3, or from q_3 to q_{10}, or from q_7 to q_8, or from q_8 to q_{11} (step 5 of $\mathcal{I}_{CS}$), there is no effect over the string representation of $CS(i)$.

  With respect to the transitions labeled with the locution $\lambda_i \in LS_{\mathcal{FOL}}$, they correspond to the continuation of dialogue in step 6 of $\mathcal{I}_{CS}$ because the agents can not accept the last contribution to the proof. After the λ_i transition is triggered the dialogue continues with the roles exchanged. The iteration in the step 4 of protocol $\mathcal{I}_{CS}$ is possible in IQ because after the transitions from state q_0 to q_2 and from state q_5 to q_7 are triggered, the string SK contains the string representation of the propositions asserted by the agent A or B respectively. Each of these propositions is removed in turn from the share knowledge when the agent playing the other role accepts them in state q_2 or q_7 respectively. Below we define δ_{IQ}:

$$\delta_{IQ}(q_0, assert_A(q \rightarrow p), \langle initial, p\rangle\alpha\{\beta\}_A\{\eta\}_B) = \\ \{(q_2, \langle assert, q \rightarrow p\rangle_A\langle contribution, q \rightarrow p\rangle_A\alpha\{(q \rightarrow p)\beta\}_A\{\eta\}_B)\}$$

$\delta_{IQ}(q_0, assert_A(\mathcal{U}), \langle challenge, p\rangle\alpha\{\beta\}_A\{\eta\}_B) = \{(q_1, \alpha\{\mathcal{U}\beta\}_A\{\eta\}_B)\}$

$\delta_{IQ}(q_0, assert_A(S), \langle challenge, p\rangle\alpha\{\beta\}_A\{\eta\}_B) =$

$\Big\{ (q_2, \langle assert, s_1\rangle_A, ..., \langle assert, s_n\rangle_A\alpha\{s_{1,A}, ..., s_{n,A}\beta\}_A\{\eta\}_B) \Big\}$ provided $S = \{s_1, ..., s_n\}$

$\delta_{IQ}(q_0, assert_A(\mathcal{U}), \langle challenge, p\rangle\alpha\{\beta\}_A\{\eta\}_B) = \{(q_1, \alpha\{\mathcal{U}\beta\}_A\{\eta\}_B\})\}$

$\delta_{IQ}(q_2, accept_B(p), \langle assert, p\rangle_A\alpha\{\beta\}_A\{\eta\}_B) = \{(q_2, \alpha\{\beta\}_A\{p\eta\}_B)\}$

$\delta_{IQ}(q_2, challenge_B(p), \langle assert, p\rangle_A\alpha\{\beta\}_A\{\eta\}_B) = \{(q_0, \langle challenge, p\rangle_B\alpha\{\beta\}_A\{\eta\}_B)\}$

$\delta_{IQ}(q_2, accept_A(q \rightarrow p), \langle contribution, q \rightarrow p\rangle_A\alpha\{\beta\}_A\{\eta\}_B) =$

$\{(q_3, \langle termination, q \rightarrow p\rangle_A\alpha\{(q \rightarrow p)\beta\}_A\{\eta\}_B)\}$

$\delta_{IQ}(q_2, \lambda_Z, \langle contribution, q \rightarrow p\rangle_A\alpha\{\beta\}_A\{\eta\}_B) = \{(q_4, \langle contribution, q \rightarrow p\rangle_A\alpha\{\beta\}_A\{\eta\}_B)\},$

$\delta_{IQ}(q_3, accept_B(q \rightarrow p), \langle termination, q \rightarrow p\rangle_A\alpha\{\beta\}_A\{\eta\}_B) = \{(q_{10}, \alpha\{\beta\}_A\{(q \rightarrow p)\eta\}_B\})\}$

$\delta_{IQ}(q_3, \lambda_Z, \langle termination, q \rightarrow p\rangle_A\alpha\{\beta\}_A\{\eta\}_B) = \{(q_4, \langle contribution, q \rightarrow p\rangle_A\alpha\{\beta\}_A\{\eta\}_B)\}$

Symmetrically:

$\delta_{IQ}(q_4, assert_B(r \rightarrow q), \langle contribution, q \rightarrow p\rangle_A\alpha\{\beta\}_A\{\eta\}_B) =$

$\{(q_5, \langle assert, r \rightarrow q\rangle_B\langle contribution, r \rightarrow q\rangle_B\alpha\{\beta\}_A\{r \rightarrow q\eta\}_B)\}$

$\delta_{IQ}(q_5, assert_B(\mathcal{U}), \langle contribution, p\rangle\alpha\{\beta\}_A\{\eta\}_B) = \{(q_1, \alpha\{\beta\}_A\{\mathcal{U}\eta\}_B)\}$

$\delta_{IQ}(q_5, assert_B(S), \langle challenge, p\rangle\alpha\{\beta\}_A\{\eta\}_B) =$

$\Big\{ (q_7, \langle assert, s_1\rangle_A, ..., \langle assert, s_n\rangle_A\alpha\{s_1, ..., s_n\beta\}_A\{\eta\}_B) \Big\}$ provided $S = \{s_1, ..., s_n\}$

$\delta_{IQ}(q_5, assert_B(\mathcal{U}), \langle challenge, p\rangle\alpha\{\beta\}_A\{\eta\}_B) = \{(q_6, \alpha\{\beta\}_A\{\mathcal{U}\eta\}_B)\}$

$\delta_{IQ}(q_7, accept_A(p), \langle assert, p\rangle_B\alpha\{\beta\}_A\{\eta\}_B) = \{(q_7, \alpha\{p\beta\}_A\{\eta\}_B)\}$

$\delta_{IQ}(q_7, challenge_A(p), \langle assert, p\rangle_B\alpha\{\beta\}_A\{\eta\}_B) = \{(q_5, \langle challenge, p\rangle_A\alpha\{\beta\}_A\{\eta\}_B)\}$

$\delta_{IQ}(q_7, accept_B(q \rightarrow p), \langle contribution, q \rightarrow p\rangle_B\alpha\{\beta\}_A\{\eta\}_B) =$

$\{(q_8, \langle termination, q \rightarrow p\rangle_B\alpha\{\beta\}_A\{(q \rightarrow p)\eta\}_B)\}$

$\delta_{IQ}(q_7, \lambda_Z, \langle contribution, q \rightarrow p\rangle_B\alpha\{\beta\}_A\{\eta\}_B) = \{(q_9, \langle contribution, q \rightarrow p\rangle_B\alpha\{\beta\}_A\{\eta\}_B)\}$

$\delta_{IQ}(q_8, accept_A(q \rightarrow p), \langle termination, q \rightarrow p\rangle_B\alpha\{\beta\}_A\{\eta\}_B) = \{(q_{11}, \alpha\{(q \rightarrow p)\beta\}_A\{\eta\}_B)\}$

$\delta_{IQ}(q_8, \lambda_Z, \langle termination, q \rightarrow p\rangle_B\alpha\{\beta\}_A\{\eta\}_B) = \{(q_0, \langle contribution, q \rightarrow p\rangle_B\alpha\{\beta\}_A\{\eta\}_B)\}$

$\delta_{IQ}(q_9, \langle assert, r \rightarrow q\rangle_A, \langle contribution, q \rightarrow p\rangle_B\alpha\{\beta\}_A\{\eta\}_B) =$

$\{(q_0, \langle assert, r \rightarrow q\rangle_A\langle contribution, r \rightarrow q\rangle_A\alpha\{(r \rightarrow q)\beta\}_A + \{\eta\}_B)\}$

- $F = \{q_1, q_6, q_{10}, q_{11}\}$.

By the way we define function δ_{IQ} and locution set $LS_{\mathcal{FOL}}$, the language of dialogues generated by $IQ \in ConvFST_2$ is equivalent to the set of dialogue moves described by the protocol $\mathcal{I}_{CS}$. A dialogue instance $\tau \in (DgArg_{\mathcal{FOL}})^+$ such that τ satisfies protocol $\mathcal{I}_{CS}$ describes any of the following situations:

1. An inquiry dialogue that finishes without solving the question when an agent A_i replies *assert*($\mathcal{U}$) to the initial inquiry topic or he replies to a locution *challenge*(p) uttered by agent A_j, $i \neq j$.

2. An inquiry dialogue that finishes solving the question when first an agent A_i accepts all the formulas that justify $q \rightarrow p$, then agent A_j accepts $q \rightarrow p$ because $\mathcal{A}(CS(j) \cup CS(i))$ includes an argument for $q \rightarrow p$ which is acceptable for him, and finally the agent A_i also accepts $q \rightarrow p$ because $\mathcal{A}(CS(j) \cup CS(i))$ includes an argument for $q \rightarrow p$ which is acceptable for him.

Equivalently, in $IQ \in ConvFST_2$ the string of locutions τ satisfying protocol $\mathcal{I}_{CS}$ is accepted iff:

1. (a) If $\tau = \alpha\ assert_A(\mathcal{U})$ then
$(q_0, \alpha\ assert_A(\mathcal{U}), \langle intial, p\rangle\{\}_A\{\}_B)$
$\vdash^* (q_1, \lambda, \delta\{\mathcal{U}\beta\}_A\{\eta\}_B)$.

   (b) If $\tau = \alpha\ challenge_A(p)assert_B(\mathcal{U})$ then
$(q_0, \alpha\ challenge_A(p)assert_B(\mathcal{U}), SK_0)$
$\vdash^* (q_7, challenge_A(p)assert_B(\mathcal{U}), \langle assert, p\rangle_B\delta\{\beta\}_A\{\eta\}_B)$
$\vdash (q_5, assert_B(\mathcal{U}), \langle challenge, p\rangle_A\delta\{\beta\}_A\{\eta\}_B)$
$\vdash (q_6, \lambda, \delta\{\beta\}_A\{\mathcal{U}\eta\}_B)$.

2. (a) If $\tau = \alpha\ accept_B(r)accept_A(p \to q)accept_B(p \to q)$ then
$(q_0, \alpha\ accept_B(r)accept_A(p \to q)accept_B(p \to q), \langle initial, x\rangle\delta\{\beta\}_A\{\eta\}_B)$
$\vdash^* \begin{pmatrix} q_2, accept_B(r)accept_A(p \to q)accept_B(p \to q), \\ \langle assert, r\rangle_A\langle contribution, p \to q\rangle_A\delta\{\beta\}_A\{\eta\}_B \end{pmatrix}$
$\vdash (q_2, accept_A(p \to q)accept_B(p \to q), \langle contribution, p \to q\rangle_A\delta\{\beta\}_A\{(r)\eta\}_B)$
$\vdash (q_3, accept_B(p \to q), \langle termination, p \to q\rangle_A\delta\{(p \to q)\beta\}_A\{(r)\eta\}_B)$
$\vdash (q_{10}, \lambda, \delta\{(p \to q)\beta\}_A\{(p \to q)(r)\eta\}_B$.

   (b) If $\tau = \alpha\ accept_A(r)accept_B(p \to q)accept_A(p \to q)$ then
$(q_0, \alpha\ accept_A(r)accept_B(p \to q)accept_A(p \to q), \langle initial, x\rangle\delta\{\beta\}_A\{\eta\}_B)$
$\vdash^* \begin{pmatrix} q_7, accept_A(r)accept_B(p \to q)accept_A(p \to q), \\ \langle assert, r\rangle_B\langle contribution, p \to q\rangle_B\delta\{\beta\}_A\{\eta\}_B \end{pmatrix}$
$\vdash (q_7, accept_B(p \to q)accept_A(p \to q), \langle contribution, p \to q\rangle_B\delta\{(r)\beta\}_A\{\eta\}_B)$
$\vdash (q_8, accept_A(p \to q), \langle termination, p \to q\rangle_B\delta\{(r)\beta\}_A\{(p \to q)\eta\}_B)$
$\vdash (q_{11}, \lambda, \delta\{(p \to q)(r)\beta\}_A\{(p \to q)\eta\}_B)$.

The interchange of locutions in IQ can be represented by the graph in figure 5.2.

Backtracking is the capacity of speakers to reply to locutions uttered at any earlier step of the dialogue and not only the previous one. The dialogue protocol $\mathcal{I} \in ConvFST_2$ is an example of the the suitability of the framework $ConvFST$ to specify dialogues with backtracking.

5.2.4 Formal Properties

With theorem 15 we provide a restriction under which the family $ConvFST_n$ of classes of languages collapses. Before proving theorem 15 we introduce the following functions, where $LS_{\mathcal{L}}$ is a set of locutions with preconditions over logic $\mathcal{L}$:

Definition 55

$$\begin{aligned} RenameLc : (LS \times Constant) &\to LS \\ RenameLc(\rho_{id_y}(\phi^{(h)}), id_x) &= \rho_{id_x}(\phi^{(h)}) \end{aligned}$$

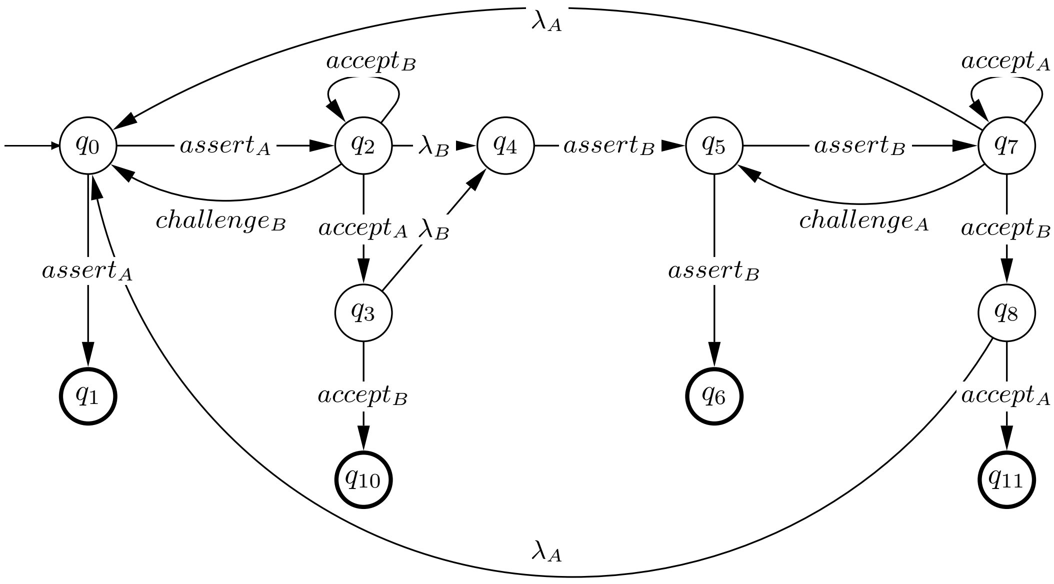

Figure 5.2: Transition graph for the interchange of locutions in $\mathcal{I}$.

Definition 56

$$\begin{aligned}
RenameDg : ((LS_{\mathcal{L}})^* \times Constant) \rightarrow\ & (LS_{\mathcal{L}})^* \\
RenameDg(\lambda, id_x) =\ & \lambda \\
RenameDg(\rho_{id_y}(\phi^{(h)})\beta, id_x) =\ & \rho_{id_x}(\phi^{(h)})\ RenameDg(\beta, id_x)
\end{aligned}$$

Definition 57

$$\begin{aligned}
RenameL : (2^{(LS_{\mathcal{L}})^*} \times Constant) \rightarrow\ & 2^{(LS_{\mathcal{L}})^*} \\
RenameL(\emptyset, id_x) =\ & \emptyset \\
RenameL(\{\alpha\} \cup L, id_x) =\ & \{RenameDg(\alpha, id_x)\} \cup RenameL(L, id_x)
\end{aligned}$$

Theorem 15

$$\left(\begin{array}{l}
\forall W \in ConvFST_n : W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{L}}, \Gamma, \delta, q_0, SK_0, F) \wedge K_{id_{n+1}} = K_{id_1} \cup ... \cup K_{id_n} \wedge \\
\left(\begin{array}{l}
\forall \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \rightarrow prec(\rho_{id_{n+1}}(v^{(h)})) \wedge \\
\left(\begin{array}{l}
\forall \rho_{id_{n+1}}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_{n+1}}(v^{(h)})) \rightarrow \\
\qquad \left(\exists \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \right)
\end{array}\right) \rightarrow \\
\qquad \left(\begin{array}{l}
\exists W' \in ConvFST_1 : RenameL(L_{Dg}(W), id_{n+1}) = L_{Dg}(W') \wedge \\
L_{SK}(W) = L_{SK}(W')
\end{array}\right)
\end{array}\right)
\end{array}\right),$$

for all $n \geq 1$.

Proof.

Considering an arbitrary $P \in ConvFST_n$ such that $P = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{L}}, \Gamma, \delta, q_0, SK_0, F)$. From P we can define $P' \in ConvFST_1$ such that $P' = (K_{id_{n+1}}, \Sigma, Q, newLS_{\mathcal{L}}, \Gamma, \delta', q_0, SK_0, F)$ where:

- $K_{id_{n+1}} = K_{id_1} \cup ... \cup K_{id_n}$,
- $newLS_{\mathcal{L}} = \{\rho_{id_{n+1}}(\phi^{(h)}) \mid \rho_{id_i}(\phi^{(h)}) \in LS_{\mathcal{L}} \wedge 1 \leq i \leq n\}$,
- $(\forall (q, \rho_{id_i}(v^{(h)}), SK) : (p, newSK) \in \delta(q, \rho_{id_i}(\phi^{(h)}), SK) \rightarrow (p, newSK) \in \delta'(q, \rho_{id_{n+1}}(\phi^{(h)}), SK))$.

We prove that $RenameL(L_{Dg}(P), id_{n+1}) = L_{Dg}(P')$ in the following way:

$$\begin{array}{l}
\beta \in RenameL(L_{Dg}(P), id_{n+1}) \\
\leftrightarrow \{\text{Definition } RenameL\} \\
(\exists \alpha : \alpha \in L_{Dg}(P) \wedge RenameDg(\alpha, id_{n+1}) = \beta) \\
\leftrightarrow \{\text{Definition } P\} \\
(q_0, \alpha, SK_0) \Rightarrow_P^* (q_f, \lambda, SK_f) \wedge q_f \in F \wedge RenameDg(\alpha, id_{n+1}) = \beta \\
\leftrightarrow \left\{ \begin{array}{l}
(\forall (q, \rho_{id_i}(\phi^{(h)}), SK) : (p, newSK) \in \delta(q, \rho_{id_i}(\phi^{(h)}), SK) \rightarrow (p, newSK) \in \delta'(q, \rho_{id_{n+1}}(\phi^{(h)}), SK)) \wedge \\
(\forall \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \rightarrow prec(\rho_{id_{n+1}}(v^{(h)}))) \wedge \\
\left(\forall \rho_{id_{n+1}}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_{n+1}}(v^{(h)})) \rightarrow \left(\exists \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)}) \right) \right) \wedge \\
K_{id_{n+1}} = K_{id_1} \cup ... \cup K_{id_n}
\end{array} \right\} \\
(q_0, \beta, SK_0) \Rightarrow_{P'}^* (q_f, \lambda, SK_f) \wedge q_f \in F \\
\leftrightarrow \{\text{Definition } P'\} \\
\beta \in L_{Dg}(P')
\end{array}$$

We prove that $L_{SK}(P) = L_{SK}(P')$ in this way:

$$\begin{array}{l}
\left(\begin{array}{l}
\forall SK \in L_{SK}(P) : (q_0, \alpha, SK_0) \Rightarrow_P^* (k, \beta, SK) \leftrightarrow \\
\qquad (q_0, RenameDg(\alpha, id_{n+1}), SK_0) \Rightarrow_{P'}^* (k, RenameDg(\beta, id_{n+1}), SK)
\end{array} \right) \\
\leftrightarrow \left\{ \begin{array}{l}
(\forall (q, \rho_{id_i}(\phi^{(h)}), SK) : (p, newSK) \in \delta(q, \rho_{id_i}(\phi^{(h)}), SK) \rightarrow (p, newSK) \in \delta'(q, \rho_{id_{n+1}}(\phi^{(h)}), SK)) \wedge \\
(\forall \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \rightarrow prec(\rho_{id_{n+1}}(v^{(h)}))) \wedge \\
\left(\forall \rho_{id_{n+1}}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_{n+1}}(v^{(h)})) \rightarrow \left(\exists \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)}) \right) \right) \wedge \\
K_{id_{n+1}} = K_{id_1} \cup ... \cup K_{id_n})
\end{array} \right\} \\
true \ \blacksquare
\end{array}$$

Below we prove a theorem that relates family $ConvFST_n$ with the Chomsky hierarchy:

Theorem 16 *$RE = EConvFST_n$, for all $n \geq 1$.*

Proof.

We prove first that $EConvFST_n \subseteq RE$. Given a dialogue protocol $W \in EConvFST_n$ defined over locution set $LS_{\mathcal{L}}$, for every locution $\rho_{id}(\phi^{(h)}) \in LS_{\mathcal{L}}$ the satisfaction of the predicate $prec(\rho_{id}(v^{(h)}))$ is computable, with $v^{(h)}$ the values of parameters $\phi^{(h)}$. Therefore $EConvFST_n \subseteq RE$.

Below we prove that $RE \subseteq EConvFST_1$. From an arbitrary one-tape Turing Machine $M = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$ we define $W \in EConvFST_1$ such that $W = (K_{id_1}, \Delta, Q, LS_{\mathcal{PL}}, \Gamma', \delta', q_0, F)$ where:

- $K_{id_1} = \emptyset$,
- Δ is an arbitrary finite set,
- $LS_{\mathcal{PL}} = \{silence_{id_1}\}$ with $prec(silence_{id_1}) = true$ and $(silence_{id_1})^* = silence_{id_1}$,
- $\Gamma' = \Gamma \cup \{\downarrow\}$, and
- δ' is defined in the following way:
  - $\delta(q,l) = (k, s, Left) \rightarrow \delta'(q, silence_{id_1}, \alpha \downarrow l\beta) = \{(k, \alpha s \downarrow \beta)\}$
  - $\delta(q,l) = (k, s, Right) \rightarrow \delta'(q, silence_{id_1}, \alpha a \downarrow l\beta) = \{(k, \alpha \downarrow as\beta)\}$
  - $\delta(q,l) = (k, s, Nill) \rightarrow \delta'(q, silence_{id_1}, \alpha \downarrow l\beta) = \{(k, \alpha \downarrow s\beta)\}$.
  - $\delta'(q, silence_{id_1}, \alpha \downarrow \beta) = \{(q, \alpha\beta)\}$, where $q \in F$.

Prove that $L(M) = L_{SK}(W) \cap \Gamma^*$ is equivalent to prove that:

$$\left[\ \forall\alpha : \quad ((q_0, \alpha) \Rightarrow^*_M (q_f, \beta) \wedge q_f \in F)) \leftrightarrow ((q_0, silence_{id_1}, \downarrow \alpha) \Rightarrow^*_W (q_f, silence_{id_1}, \beta)))\ \right]$$

By the way we defined function δ' it is clear that the predicate above is true. ■

From theorem 16 and the proof that *EConvEREG* systems have the expressiveness of Turing Machines, we get the following corollary:

Corollary 6 *$EConvFST_n = EConvEREG_m$, for all $n, m \geq 1$.*

5.2.5 Applicability in Dialogue Theory

Family $ConvFST_n$ allows to specify dialogue protocols with the following features:

- The number of conversants is n.
- The agents can not learn or modify their own knowledge bases but they can change the shared knowledge base through the dialogue.
- Backtracking, turn-taking and arbitrary replying policies can be specified.
- Arbitrary locution sets, agent knowledge bases, agent reasoning strategies and dialogue initiatives can be used.

A way to decrease the expressive power of *ConvFST* systems is to define subclasses of *ConvFST* that can be applicable in Dialogue Theory and exhibit good properties. We introduce three subclasses of *ConvFST*:

1. *Conditional Finite State Transition* (*CondFST*) systems. In this framework the agents are provided with a finite set of conditional rules that they use to decide their participation in the dialogue. The agents use these rules to decide how to modify the string corresponding to the shared knowledge depending on the presence and\or absence of substrings(information) in that string. When in *CondFST* systems we consider the share knowledge generated and non erasing rewritting rules we get a subclass of CS grammars.

2. *Limited memory Finite State Transition* ($lmFST$) systems. In this framework the shared knowledge is limited to at most the last uttered locution. This class has the same expressive power as CS grammars when we consider the share knowledge generated, so the membership problem is decidable.

3. *Regularly Controlled Finite State Transition (rCFST)* systems. In this framework the locutions have no parameters and each locution is associated with a non-erasing CF rule. The precondition of every locution evaluates true if the associated CF rule can be applied over the string of shared knowledge. When a locution is uttered a new string of shared knowledge is obtained from the application of the CF rewriting rule associated with that locution. This framework has the same expressive power as *regularly Controlled* Grammar systems. Therefore $rCFST$ systems correspond to a type of mildly CS grammar so the membership problem is decidable.

5.3 $CondFST$ systems

We introduce *Conditional Finite State Transition* ($CondFST$) systems with degree $n \geq 1$ and condition $c \in \{b, s\}$, which differ from $ConvFST$ systems on the following restrictions:

- The agent private knowledge bases are given by a finite set of conditional rules of the type $(e, f, \rho_{id}(\phi^{(m)}) : P)$, where Γ is the finite set of symbols used in the string of shared knowledge, $e \in \Gamma^*$ is a permitting context, $f \in \Gamma^*$ is a forbidding context, $\rho_{id}(\phi^{(m)})$ is a locution and (Γ, P) is a 0L component scheme.

- Before a speaker A_{id} can utter a locution $\rho_{id}(v^{(m)})$ in a conversation with shared knowledge $SK \in \Gamma^*$, it has to inspect his knowledge base K_{id} looking for a rule $(e, f, \rho_{id}(\phi^{(m)}) : P)$ that satisfies the following:

  - If $c = b$ then SK has to contain the block of information e, i.e. $SK = \alpha e \beta$ and SK can not contain the block of data f, what means $\neg(\exists \alpha, \beta : SK = \alpha f \beta)$. If these conditions are satisfied then agent A_{id} can utter locution $\rho_{id}(v^{(m)})$. The consequence of uttering $\rho_{id}(v^{(m)})$ is the application of rules P over SK.
  - If $c = s$ then SK has to contain the information $e = e_0...e_t$ scattered in SK ($SK = \alpha_1 e_1 \alpha_2 ... e_t \alpha_{t+1}$) and SK can not contain data $f = f_1...f_q$ scattered in SK ($SK \neq \beta_1 f_1 \beta_2 ... f_q \beta_{q+1}$). Only if these conditions are satisfied then the agent A_{id} can utter locution $\rho_{id}(v^{(m)})$. The consequence of uttering $\rho_{id}(v^{(m)})$ is the application of rules P over SK.

5.3.1 Formal Definition

Below we formally define *Conditional Finite State Transition* ($CondFST$) systems with degree $n \geq 1$ and condition $c \in \{b, s\}$, interpreting the conditional predicate π_c as explained in definition 33.

Definition 58

$$CondFST_n = \left\{ \begin{array}{l} W \in ConvFST_n \mid W = (K_{id_1},...,K_{id_n},\Sigma,Q,LS_{\mathcal{FOL}},\Gamma,\delta,q_0,SK_0,F)\wedge \\ \Sigma = \Gamma \cup LS_{\mathcal{FOL}} \cup \{(,),:,\rightarrow\}\wedge \\ K_{id_i} = \left\{ \begin{array}{l} (e,f,\rho_{id_i}(\phi^{(m)}):P) \mid e\in\Gamma^* \wedge f\in\Gamma^*\wedge \rho_{id_i}(\phi^{(m)})\in LS_{\mathcal{FOL}} \\ \wedge(\Gamma,P) \textit{ is a } 0L \textit{ component scheme}\} \end{array}\right\}\wedge \\ \left(\begin{array}{l} \forall\rho_{id_i}(\phi^{(m)})\in LS_{\mathcal{FOL}},\ q,k\in Q,\ SK,newSK\in\Gamma^*: \\ prec(\rho_{id_i}(\phi^{(m)})) = \left(\begin{array}{l} \exists(e,f,\rho_{id_i}(\phi^{(m)}):P)\in K_{id_i}: \pi_c(e,SK)=1\wedge \\ \pi_c(f,SK)=0\end{array}\right)\end{array}\right)\wedge \\ \left(\begin{array}{l} \forall q,k\in Q,\ SK,newSK\in\Gamma^*,\ \rho_{id_i}(\nu^{(m)})\alpha\in(LS_{\mathcal{FOL}})^+: \\ \left(\begin{array}{l}\exists(e,f,\rho_{id_i}(\phi^{(m)}):P)\in K_{id_i}: prec(\rho_{id_i}(\nu^{(m)}))\wedge \\ SK\Rightarrow^P newSK\end{array}\right)\rightarrow \\ \qquad\qquad (q,\rho_{id_i}(\nu^{(m)})\alpha,SK)\Rightarrow_W (k,\alpha,newSK) \\ \\ \end{array}\right) \end{array}\right\}$$

We denote by $CondFST_n(i,j,c)$, $n \geq 1$, $c \in \{b,s\}$, $i,j \geq 0$, the family of languages $L_{SK}(W)$ where $W \in CondFST_n$, all the permitting contexts have length at most i, all the forbidden context have length at most j and predicate π_c is used to verify the presence or absence of contexts. When the number of agents, the length of the permitting context, or the length of the forbidding contexts is not bounded, then we replace the corresponding parameter with ∞.

When we are interested only in strings over some alphabet T, hence in the language $L_{SK}(W) \cap T^*$, then we speak about an *Extended Conditional Finite State Transition (ECondFST)* system. We denote by $ECondFST_n(i,j,c)$, the family of languages $L_{SK}(W) \cap T^*$ where $W \in CondFST_n(i,j,c)$.

5.3.2 Formal Properties

The following basic relations directly follow from the definition of $ECondFST_n(i,j,c)$:

Lemma 2

1. $ECondFST_n(i,j,c) \subseteq ECondFST_m(i',j',c)$, *for all* $c \in \{b,s\}$, $1 \leq n \leq m$ *and* $i \leq i'$, $j \leq j'$.
2. $ECondFST_n(i,j,b) \subseteq ECondFST_n(i,j,s)$, *for all* $n \geq 1$ *and* $i,j \in \{0,1\}$.
3. $ECondFST_n(0,0,c) = ETOL, n \geq 1$, $c \in \{b,s\}$.

With the theorem below we introduce some restrictions over the preconditions of the locutions in order to make $ECondFST_n$ collapse into $ECondFST_1$:

Theorem 17

$$\left(\begin{array}{l} \forall W \in ECondFST_n : W = (K_{id_1},...,K_{id_n},\Sigma,Q,LS_{\mathcal{L}},\Gamma,\delta,q_0,SK_0,F) \wedge K_{id_{n+1}} = K_{id_1} \cup ... \cup K_{id_n} \wedge \\ \left(\exists W' \in ECondFST_1 : RenameL(L_{Dg}(W), id_{n+1}) = L_{Dg}(W') \wedge L_{SK}(W) = L_{SK}(W') \right) \end{array} \right),$$

for all $n \geq 1$.

Proof. We consider an arbitrary dialogue protocol $W \in ECondFST_n$ such that

$$W = (K_{id_1},...,K_{id_n},\Sigma,Q,LS_{\mathcal{FOL}},\Gamma,\delta',q_0,SK_0,F)$$

From W we define a dialogue protocol $W' \in ECondFST_1$,

$$W' = (K_{id_{n+1}}, \Sigma, Q, newLS_{\mathcal{FOL}}, \Gamma, \delta, q_0, SK_0, F) \text{ such that:}$$

- $K_{id_{n+1}} = \{(d, e, \rho_{id_{n+1}}(\phi^{(h)}) : P) \mid (d, e, \rho_{id_i}(\phi^{(h)}) : P) \in K_{id_i} \wedge 1 \leq i \leq n\}$,
- $newLS_{\mathcal{FOL}} = \{\rho_{id_{n+1}}(\phi^{(h)}) \mid \rho_{id_i}(\phi^{(h)}) \in LS_{\mathcal{FOL}} \wedge 1 \leq i \leq n\}$,
- $(\forall (q, \rho_{id_i}(\phi^{(h)}), SK) : (p, newSK) \in \delta(q, \rho_{id_i}(\phi^{(h)}), SK) \rightarrow (p, newSK) \in \delta'(q, \rho_{id_{n+1}}(\phi^{(h)}), SK))$.

First we need a supplementary proof. We need to prove the following:

$$(\forall \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \rightarrow prec(\rho_{id_{n+1}}(v^{(h)}))) \wedge$$
$$\left(\begin{array}{l} \forall \rho_{id_{n+1}}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_{n+1}}(v^{(h)})) \rightarrow \\ \qquad\qquad \left(\exists \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \right) \end{array} \right)$$

We prove it for an arbitrary locution $\rho_{id_i}(v^{(h)})$:

$prec(\rho_{id_i}(v^{(h)}))$
$\leftrightarrow$ {Definition *ECondFST* systems and definition function *RenameLc*}
$$\left(\begin{array}{l} \exists (e, f, \rho_{id_i}(\phi^{(m)}) : P) \in K_{id_i} : \pi_c(e, SK) = 1 \wedge \pi_c(f, SK) = 0 \wedge \\ RenameLc(\rho_{id_i}(\phi^{(m)}), id_{n+1}) = \rho_{id_{n+1}}(\phi^{(m)}) \end{array} \right)$$
$\leftrightarrow$ {Definition W'}
$$\left(\begin{array}{l} \exists (e, f, \rho_{id_{n+1}}(\phi^{(m)}) : P) \in K_{id_{n+1}} : \pi_c(e, SK) = 1 \wedge \pi_c(f, SK) = 0 \wedge \\ RenameLc(\rho_{id_i}(\phi^{(m)}), id_{n+1}) = \rho_{id_{n+1}}(\phi^{(m)}) \end{array} \right)$$
$\leftrightarrow$ {Definition *ECondFST* systems and definition function *RenameLc*}
$prec(\rho_{id_{n+1}}(v^{(h)}))$.

The proof continues as explained in theorem 15. ■

The result below connects *ECondFST* systems with the *Extended Conditional Tabled Eco-Grammar (ECTEG)* systems introduced in Chapter 2.

Theorem 18 *$ECTEG_n(i, j, c) \subseteq ECondFST_1(\infty, \infty, c)$, for all $n \geq 1$, $i, j \geq 0$, $c \in \{b, s\}$.*

Proof. In [Mih94] it was proved that $ECTEG_n(i, j, c) = ECTEG_1(i, j, c)$ for arbitrary $n \geq 1$, $i, j \geq 0$ and $c \in \{b, s\}$. Then we can prove this theorem replacing n for 1.

First we prove it for the scattered condition: given an arbitrary $W \in ECTEG_1(i, j, s)$ such that $W = (E, A_1)$ with:

- $E = (V_E, Rules_E)$,
- $Rules_E = (e_1, f_1 : P_1), ..., (e_k, f_k : P_k)$,
- $A_1 = (V_1, Rules_1)$,
- $Rules_1 = (g_1, h_1 : R_1), ..., (g_s, h_s : R_s)$,
- $V_E \cap V_1 = \emptyset$.

From W we can define $W' \in ECondFST_1(2i, 2j, s)$ such that

$$W' = (K_{id}, \Sigma, Q, LS_{FOL}, \Gamma, \delta, q_0, SK_0, F) \text{ with:}$$

- $Q = \{q_0\}$,
- $LS_{FOL} = \{silence_{id}\}$ and $silence_{id} = (silence_{id})^*$,
- $F = \emptyset$,
- $\Gamma = V_E \cup V_1$,
- $SK_0 = (w_{E,0}, w_{1,0})$, and
- $K_{id} = \{(eg, fh, silence_{id} : P \cup R) \mid (e, f : P) \in Rules_E \wedge (g, h : R) \in Rules_1\}$.

It is enough to prove the following:

$$\left(\begin{array}{l} \forall k \geq 0 : (w_{E,k}, w_{1,k}) \Rightarrow_W (w_{E,k+1}, w_{1,k+1}) \leftrightarrow \\ \qquad (q_0, silence_{id}, (w_{E,k}, w_{1,k})) \Rightarrow_{W'} (q_0, silence_{id}, (w_{E,k+1}, w_{1,k+1})) \end{array} \right).$$

We prove it below for an arbitrary $k \geq 0$:

$(w_{E,k}, w_{1,k}) \Rightarrow_W (w_{E,k+1}, w_{1,k+1})$
$\leftrightarrow$ { Definition $W \in ECTEG_1(i, j, s)$}
$(\exists (e, f : P) \in Rules_E : \pi_s(e, w_{1,k}) = 1 \wedge \pi_s(f, w_{1,k}) = 0 \wedge w_{E,k} \Rightarrow^P w_{E,k+1}) \wedge$
$(\exists (g, h : R) \in Rules_1 : \pi_s(g, w_{E,k}) = 1 \wedge \pi_s(h, w_{E,k}) = 0) \wedge w_{1,k} \Rightarrow^R w_{1,k+1})$
$\leftrightarrow$ {Definition $K_{id} \wedge V_E \cap V_1 = \emptyset$}
$\left(\begin{array}{l} \exists (eg, fh, silence_{id} : P \cup R) \in K_{id} : \pi_s(eg, (w_{E,k}, w_{1,k})) = 1 \wedge \\ \pi_s(fh, (w_{E,k}, w_{1,k})) = 0 \wedge w_{E,k} \Rightarrow^P w_{E,k+1} \wedge w_{1,k} \Rightarrow^R w_{1,k+1} \end{array} \right)$
$\leftrightarrow$ { Definition $W' \in ECondFST_1(2i, 2j, s)$}
$(q_0, silence_{id}, (w_{E,k}, w_{1,k})) \Rightarrow_{W'} (q_0, silence_{id}, (w_{E,k+1}, w_{1,k+1})))$.

When we consider a block condition the proof consists on considering an arbitrary $W \in ECTEG_1(i, j, b)$. From W we can define $W' \in ECondFST_1(\infty, \infty, b)$ similar to the previous case, only differing in the following:

$$K_{id} = \{(e\alpha g, f\beta h, silence_{id} : P \cup R) \mid (e, f : P) \in Rules_E \wedge (g, h : R) \in Rules_1 \wedge \alpha, \beta \in \Gamma^*\}$$

Then we need to prove the following for an arbitrary $k \geq 0$:

$(w_{E,k}, w_{1,k}) \Rightarrow_W (w_{E,k+1}, w_{1,k+1})$
$\leftrightarrow$ { Definition $W \in ECTEG_1(i, j, b)$}
$(\exists (e, f : P) \in Rules_E : \pi_b(e, w_{1,k}) = 1 \wedge \pi_b(f, w_{1,k}) = 0 \wedge w_{E,k} \Rightarrow^P w_{E,k+1}) \wedge$
$(\exists (g, h : R) \in Rules_1 : \pi_b(g, w_{E,k}) = 1 \wedge \pi_b(h, w_{E,k}) = 0) \wedge w_{1,k} \Rightarrow^R w_{1,k+1})$
$\leftrightarrow$ {Definition $K_{id} \wedge V_E \cap V_1 = \emptyset$}
$\left(\begin{array}{l} \exists (e\alpha g, f\beta h, silence_{id} : P \cup R) \in K_{id} : \pi_b(e\alpha g, (w_{E,k}, w_{1,k})) = 1 \wedge \\ \pi_b(f\beta h, (w_{E,k}, w_{1,k})) = 0 \wedge w_{E,k} \Rightarrow^P w_{E,k+1} \wedge w_{1,k} \Rightarrow^R w_{1,k+1} \end{array} \right)$
$\leftrightarrow$ { Definition $W' \in ECondFST_1(\infty, \infty, b)$}
$(q_0, silence_{id}, (w_{E,k}, w_{1,k})) \Rightarrow_{W'} (q_0, silence_{id}, (w_{E,k+1}, w_{1,k+1})))$. ■

Theorem 19 connects *ECondFST* systems with non erasing rewriting rules with CS grammars in the Chomsky hierarchy:

Theorem 19 *$ECondFST_{e,n}(i, j, c) \subseteq CS$, for all $n \geq 1$, $i, j \geq 0$, $c \in \{b, s\}$.*

Proof.

With Theorem 12 it was proved that $ECondEG_{e,n}(i, j, c) \subseteq CS$, for all $n \geq 1$, $i, j \geq 0$, $c \in \{b, s\}$. From an arbitrary dialogue protocol $W \in ECondFST_{e,n}(i, j, c)$ defined as

$$W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{FOL}}, \Gamma, \delta, q_0, SK_0, F),$$

we can construct a $ECondEG_{e,n}(i, j, c)$ system $W' = ((\Gamma, \emptyset), A_1, .., A_n)$ such that for all $i \geq 1$: $A_i = (\emptyset, \emptyset, P_{id_i})$ and $P_{id_i} = \{(e, f : P) \mid (e, f, \rho_{id_i}(\phi^{(m)}) : P) \in K_{id_i}\}$.

Clearly $L_{SK}(W) = L_E(W', \sigma_0)$ for all σ_0. ■

5.3.3 Applicability in Dialogue Theory

Family $CondFST_n$ allows to specify dialogue protocols with the following features:

- The number of conversants is n
- The agents can not learn or modify their own knowledge bases but they can modify the shared knowledge base through the dialogue.
- Backtracking, turn-taking and arbitrary replying policies can be implemented.
- Arbitrary locution sets are allowed.
- Agents private knowledge bases are limited to finite sets of conditional rules of the type $(e, f, \rho_{id}(\phi^{(m)}) : P)$, where Γ is the finite set of symbols used in the string of shared knowledge, $e \in \Gamma^*$ is a permitting context, $f \in \Gamma^*$ is a forbidding context, $\rho_{id}(\phi^{(m)})$ is a locution and (Γ, P) is a 0L component scheme.
- The reasoning strategy of an agent is fixed and corresponds to find a conditional rule from its private knowledge base that can be applied over the shared knowledge.
- Arbitrary dialogue initiatives can be chosen.

5.4 *lmFST* systems

ConvEREG and *ConvFST* systems provide a common string that every active agent can access to, which constitutes what in pragmatics is called a *dialogue context*. A dialogue context is a common repository of agents' shared knowledge where agents can make visible those communication actions that constitute their interface to the other agents (observable behavior). The notion of dialogue context was introduced in *Linguistics* but it has already been considered in Dialogue Theory with names like agent observable behaviors [VO02], histories of communication [EBHM03], traces of communication [WGS87] or social semantics [PWA03b].

In section 5.2.3 we explained the role of the concept of social semantics in the argumentation-based dialogue protocols from [PWA03b]: an agent X can construct its arguments from its private knowledge base K_X but also form the set of commitments $CS(Y)$ from agent Y. The principle behind social semantics is that when agents utter locutions they state publicly their knowledge, which is saved in a commitment store. The truth of an agent's speech acts can never be fully verified [Woo00], but at least an agent's consistency can be assessed inspecting the content of the commitment store. Then within an argumentation-based dialogue system the reasons supporting these expressions can be analyzed leading to an acceptance, rejection, challenge or contra-argumentation from other agents.

In this section we introduce a type of *ConvFST* system that we call *limited memory Finite State Transition (lmFST)* system where we replace the string of shared knowledge for a memory limited to at most the last uttered locution.

In [PWA03b] it was proved that in the argumentation-based framework that we presented in section 5.2.3 agent decision procedures are computable. Therefore we can use the argumentation-based framework from [PWA03b] to provide an example showing a practical implication of replacing a string for a memory limited to at most the last uttered locution in dialogue protocols where the private knowledge bases of the agents are fixed and cannot be modified during run-time.

Example 3 *Let us consider an information-seeking protocol between agents A and B provided respectively with argumentation systems $\langle \Sigma_A, Undercut_A, Pref_A\rangle$ and $\langle \Sigma_B, Undercut_B, Pref_B\rangle$ as explained in section 5.2.3. Such that $\Sigma_A = \{\neg p\}$ and $\Sigma_B = (\{p, \neg r, p \rightarrow r, \neg r \rightarrow p\}$ with p, r formulas in propositional logic. Both agents use propositional logic $\mathcal{PL}$ to prove that a proposition is inferred from their knowledge bases and they are provided with a shared memory S initialized with λ.*

We choose this set of locutions:

$$LS_{\mathcal{FOL}} = \left\{ \begin{array}{l} question_i(p), challenge_i(T), assert_i(T) \mid p \text{ is a formula in } \mathcal{PL} \wedge \\ T \text{ is a set of formulas in } \mathcal{PL} \wedge i \in \{A, B\} \end{array} \right\},$$

with the following preconditions and postconditions that we stay in natural language:

- *Agent A asks $question_A(p)$ at the beginning of the dialogue iff $S = \lambda$ and it can not deduce p from its argumentation system and it places the locution in S.*

- *Agent A utters $challenge_A(P)$ iff $S = assert_B(T)$, $T = P \cup R$, and for all the propositions p in P it cannot deduce p from its argumentation system. Then it replaces the content of S for $challenge_A(P)$.*

- *Agent B utters $assert_B(\{p\})$ iff $S = question_A(p)$ and p is not an axiom in its argumentation system and it can deduce p from its argumentation system. The agent replaces the content of S for $assert_B(\{p\})$.*

- *Agent B utters $assert_B(\{\mathcal{U}\})$ iff $S = question_A(p)$ and p is an axiom in its argumentation system or it cannot deduce p, or $S = challenge_A(T)$ and any proposition t in T is an axiom in its argumentation system or it cannot deduce t. The agent replaces the content of S for $assert_B(\{\mathcal{U}\})$.*

- *Agent B utters $assert_B(P)$ iff $S = challenge_A(T)$, none proposition in T is an axiom in its argumentation system and it can deduce that the set of propositions P provide arguments for T. The agent replaces the content of S for $assert_B(P)$.*

Using the set of locutions $LS_{\mathcal{FOL}}$ and the shared memory S of at most one locution we can obtain the following dialogue instance corresponding to an information-seeking protocol:

1. $S = \lambda$ and agent A utters $question_A(r)$.
2. $S = question_A(r)$ and agent B answers $assert_B(\{r\})$, because it can support r with arguments $\{p, p \to r\}$.
3. $S = assert_B(\{r\})$ and agent A utters $challenge_A(\{r\})$ because it cannot prove r from $\{\neg p\}$.
4. $S = challenge_A(\{r\})$ and agent B answers $assert_B(\{p \to r, p\})$. .
5. $S = assert_B(\{p \to r, p\})$ and agent A replies with $challenge_A(\{p\})$ because it cannot deduce $\neg(p \to r)$ from $\{\neg p\}$.
6. $S = challenge_A(\{p\})$ and agent B says $assert_B(\{\neg r, \neg r \to p\})$.
7. $S = assert_B(\{\neg r, \neg r \to p\})$ and agent A utters $challenge_A(\{\neg r \to p\})$ because it cannot deduce r from $\{\neg p\}$.
8. $S = challenge_A(\{\neg r \to p\})$ and agent A utters $challenge_A(\emptyset)$ because it cannot deduce $\neg(\neg r \to p)$ from $\{\neg p\}$.
The dialogue finishes successfully. There are no propositions to be challenged.

Let us consider now that agents A and B are respectively provided with argumentation systems $\langle \Sigma_A \cup CS(B), Undercut_A, Pref_A\rangle$ and $\langle \Sigma_B \cup CS(A), Undercut_B, Pref_B\rangle$, and with commitment stores $CS(A)$ and $CS(B)$ as explained in section 5.2.3. The inference systems Σ_A, Σ_B are the same as in previous case. Now instead of a shared memory S of at most one locution the agents can use a string SK as shared memory. The agents use part of SK as a shared memory S' of at most one locution and the rest as commitment stores $CS(A)$ and $CS(B)$. SK is initialized with empty memory and empty commitment stores, i.e. $SK = \lambda$.

We use the same set of locutions $LS_{\mathcal{FOL}}$ but we add to the postconditions the effect that the action of uttering locutions has over the commitment stores $CS(A)$ and $CS(B)$:

- *Agent A asks $question_A(p)$ at the beginning of the dialogue iff the memory S' is empty and it cannot deduce p from its argumentation system and it places the locution in S'.*

- *Agent A utters $challenge_A(P)$ iff $S' = [assert_B(T)]$, $T = P \cup R$, and for all p in P it cannot deduce p from its argumentation system. Then it replaces the content of S' for $challenge_A(P)$.*

- *Agent B utters $assert_B(\{p\})$ iff $S' = [question_A(p)]$ and p is not an axiom in its argumentation system and it can deduce p from its argumentation system. The agent replaces the content of S' for $assert_B(\{p\})$ and adds p to $CS(B)$.*

- *Agent B utters $assert_B(\{\mathcal{U}\})$ iff $S' = [question_A(p)]$ and p is an axiom or it cannot deduce p, or $S' = [challenge_A(P)]$ and for some proposition p in P the proposition p is an axiom in its argumentation system or it cannot deduce p. The agent replaces the content of S' for $assert_B(\{\mathcal{U}\})$.*

- *Agent B utters* $assert_B(P)$ *iff* $S' = [challenge_A(T)]$ *and no proposition in* T *is an axiom in its argumentation system and it can deduce that the set of propositions* P *provide arguments for* T*. The agent replaces the content of* S' *for* $assert_B(P)$ *and adds* p *to* $CS(B)$*.*

Using the set of locutions $LS_{\mathcal{PL}}$*, a memory* S' *of at most one locution and commitment stores* $CS(A)$ *and* $CS(B)$ *we can obtain the following dialogue instance corresponding to an information-seeking protocol:*

1. $S' = []$ *and* $CS(A) = CS(B) = \emptyset$*. Agent A utters* $question_A(r)$*.*

2. $S' = [question_A(r)]$ *and* $CS(A) = CS(B) = \emptyset$*. Agent B answers* $assert_B(\{r\})$*, because it can prove r from* $\{p, p \to r\}$

3. $S' = [assert_B(\{r\})]$*,* $CS(A) = \emptyset$ *and* $CS(B) = \{r\}$*. Agent A utters* $challenge_A(\{r\})$*, because it cannot prove r from* $\{\neg p\} \cup \emptyset$*.*

4. $S' = [challenge_A(\{r\})]$*,* $CS(A) = \emptyset$ *and* $CS(B) = \{r\}$*. Agent B answers* $assert_B(\{p \to r, p\})$*.*

5. $S' = [assert_B(\{p \to r, p\})]$*,* $CS(A) = \emptyset$ *and* $CS(B) = \{r, p \to r, p\}$*. Agent A replies with* $challenge_A(\{p\})$ *because it cannot deduce* $\neg(p \to r)$ *from* $\{\neg p\} \cup \{r, p \to r, p\}$*.*

6. $S' = [challenge_A(\{p\})]$*,* $CS(A) = \{p \to r\}$ *and* $CS(A) = \{r, p \to r, p\}$*. Agent B replies with* $assert_B(\{\neg r, \neg r \to p\})$*.*

7. $S' = [assert_B(\{\neg r, \neg r \to p\})]$*,* $CS(A) = \{p \to r\}$ *and* $CS(B) = \{r, p \to r, p, \neg r, \neg r \to p\}$*. Agent A utters* $challenge_A(\{\neg r\})$ *because it can deduce r from* $\{\neg p\} \cup \{r, p \to r, p, \neg r, \neg r \to p\}$*.*

8. $S' = [challenge_A(\{\neg r\})]$*,* $CS(A) = \{p \to r\}$ *and* $CS(B) = \{r, p \to r, p, \neg r, \neg r \to p\}$*. Because agent B has the axiom* $\neg r$ *it replies* $assert_B(\{\mathcal{U}\})$*.*

The dialogue finishes unsuccessfully, agent B cannot prove to agent A that r is true, because due to the consideration of a commitment store $CS(B)$ *agent A can detect an inconsistency in the knowledge base of agent B: agent B first* asserts *r and then it* asserts $\neg r$ *during the dialogue.*

As the example above shows in the case of argumentation-based dialogues the restriction of the string of shared knowledge to the last uttered locution can reduce the capacity of the agents to detect inconsistencies between the arguments interchanged during the dialogue, leading to wrong inferences.

In [GGF08] we worked on the comparison of concurrent processes defined in the framework of Coloured Petri Nets, focusing only on the labeled transition systems generated and abstracting from the agent reasonings strategies. We based our study on the notions of simulation equivalence for the process algebra called pi-calculus [Mil99]. Examples of simulation equivalences that we considered are branching time similarity, observable similarity, strong similarity, week similarity, etc. Our intention was to recognize processes that were bisimilar to be able to interchange them. But as this example shows, a deeper comparison of dialogues where the rationality of the agents is considered should be studied.

5.4.1 Formal Definition

Definition 59 *A dialogue protocol* $W \in lmFST_n$ *of degree* $n \geq 1$ *is a tuple:*

$$W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{L}}, \Gamma, \delta, q_0, Z_0, F)$$

where:

- *Σ, K_{id_i}, for all $1 \leq i \leq n$, Q, q_0 and F are defined as in $ConvFST$ systems;*
- *$LS_{\mathcal{L}}$ is a finite set of locutions. Considering $SK \in LS_{\mathcal{L}} \cup \{\lambda\}$ as the memory, we have:*
  $LS_{\mathcal{L}} = \left\{ \rho_{id_i}(\phi^{(m)}) \mid m \geq 0 \wedge 1 \leq i \leq n \wedge prec(\rho_{id_i}(\phi^{(m)}))\ \textit{is a wff in logic}\ \mathcal{L}\} \right\}$

  The locution $\rho_{id_i}(\phi^{(m)}) \in LS_{\mathcal{L}}$ with constants ρ, id_i and terms $\phi^{(1)}, ..., \phi^{(m)}$ has associated a well formulated formula $prec(\rho_{id_i}(\phi^{(m)}))$ in logic $\mathcal{L}$. We call to $prec(\rho_{id_i}(\phi^{(m)}))$ the precondition of locution $\rho_{id_i}(\phi^{(m)})$. For every locution $\rho_{id_i}(v^{(m)}) \in LS_{\mathcal{L}}$ with parameters values $v^{(m)}$ checking if $prec(\rho_{id_i}(v^{(m)}))$ evaluates true is decidable. There cannot be a locution $\rho_{id_i}(\phi^{(m)}) \in LS_{\mathcal{L}}$ with more than one precondition.
- *$\Gamma = LS_{\mathcal{L}} \cup \{\lambda\}$ is the memory alphabet and $LS_{\mathcal{L}} \cap \{\lambda\} = \emptyset$;*
- *$Z_0 = \lambda$ is the initial memory symbol, and*
- *δ is a finite transition relation $(Q \times LS_{\mathcal{L}} \times \Gamma) \rightarrow 2^{Q \times \Gamma}$.*

Considering that a sequence of locutions $\rho_i(v^{(m)})\beta \in (LS_{\mathcal{L}})^+$ with parameter values $v^{(m)}$ is being processed, for any $q \in Q$, $\rho_i(v^{(m)}) \in LS_{\mathcal{L}}$, $\beta \in (LS_{\mathcal{L}})^*$, the memory content $\tau \in \Gamma$, the interpretation of

$$\delta(q, \rho_i(v^{(m)}), \tau) = \{(p_1, \gamma_1), ..., (p_m, \gamma_m)\}$$

is that if the dialogue protocol W is in state q with current locution $\rho_i(v^{(m)})$, with memory content τ, and $prec(\rho_i(v^{(m)}))$ evaluates true, then W can for any $1 \leq x \leq m$ replace τ with $\gamma_x = \rho_i(v^{(m)})$ or $\gamma_x = \lambda$, take the first locution in β and enter state p_x.

To formally describe the configuration of a dialogue protocol W at a given instant we define what we call an *instantaneous description*. An instantaneous description records the state, the sequence of unprocessed locutions and the content of the memory:

Definition 60 *An instantaneous description for a dialogue protocol is a tuple (q, α, γ) where $q \in Q$ is the current state, $\alpha \in (LS_{\mathcal{L}})^*$ is the remaining sequence of locutions, $\gamma \in \Gamma$ is the content of the memory.*

The definitions of the relation $\vdash$, the language of dialogues generated and the language of share knowledge generated is inherited from $ConvFST$ systems.

We denote by $lmFST_n$, $n \geq 1$, the family of languages $L_{SK}(W)$ where $W \in lmFST_n$. When the number of agents is not bounded, then we replace the corresponding parameter with ∞.

When we are interested only in strings over some alphabet T, hence in the language $L_{SK}(W) \cap T^*$, then we speak about *Extended limited memory Finite State Transition (ElmFST)* systems. We denote by $ElmFST_n$, the family of languages $L_{SK}(W) \cap T^*$ where $W \in lmFST_n$.

5.4.2 Formal Properties

With theorem 20 we provide a restriction under which the family $lmFST_n$ of classes of languages collapses into the family $lmFST_1$.

Theorem 20

$$\left(\begin{array}{l} \forall W \in lmFST_n : W = (K_{id_1},...,K_{id_n},\Sigma,Q,LS_{\mathcal{L}},\Gamma,\delta,q_0,Z_0,F) \wedge K_{id_{n+1}} = K_{id_1}\cup ...\cup K_{id_n}\wedge \\ \left(\begin{array}{l} (\forall\rho_{id_i}(v^{(h)})\in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \rightarrow prec(\rho_{id_{n+1}}(v^{(h)})))\wedge \\ \left(\forall \rho_{id_{n+1}}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_{n+1}}(v^{(h)})) \rightarrow \left(\exists\rho_{id_i}(v^{(h)})\in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)}))\right)\right)\rightarrow \\ \qquad \left(\exists W'\in lmFST_1 : RenameL(L_{Dg}(W),id_{n+1}) = L_{Dg}(W')\wedge L_{SK}(W)=L_{SK}(W')\right)\end{array}\right)\end{array}\right),$$

for all $n \geq 1$.

Proof. Similar proof to the one provided for theorem 15. ■

Below we prove a result that connects $ElmFST$ systems with the Chomsky hierarchy:

Theorem 21 $ElmFST_n = CS$*, for all* $n \geq 1$.

Proof. We prove first that $ElmFST \subseteq CS$. Given an arbitrary dialogue protocol $W \in ElmFST_n$, $W = (K_{id_1},...,K_{id_n},\Sigma,Q,LS_{\mathcal{FOL}},\Gamma,\delta,q_0,Z_0,F)$ then from any $\zeta \in LS^*_{\mathcal{F}}$ and $GMemory \in LS_{\mathcal{F}} \cup \lambda$ we construct a Linear-Bounded automata W' with two tracks.

The firs track of W' is the input tape. We initialize the second track of W' with the string $\dagger(q_0,\zeta,Z_0X^t)\dagger$, where the symbol $\dagger$ is used to delimit the size of the tape and t is the size of the longest locution in ζ. We define W' to simulate W such that if $(q,\alpha,Memory) \Rightarrow_W (k,\beta,newMemory)$ then in $b \geq 0$ derivation steps the content of first tape $\dagger(q,\alpha,Memory\ X^r)\dagger$ is replaced for $\dagger(k,\beta,newMemory\ X^q)\dagger$ with $r,q \leq t$. W' halts when the content of the tape is $\dagger(f,\lambda,GMemory\ X^e)\dagger$ with $f \in F$ and $e \leq t$.

To prove that $CS \subseteq ElmFST_n$ we consider an arbitrary one-sided normal form CS grammar $G = (N,T,P,S)$ where every rule has one of the following shapes: $A \rightarrow BC$, $A \rightarrow a$, $AB \rightarrow AC$ for some $A,B,C \in N$ and $a \in T$. From G we can construct $D \in ElmFST_1$, $D = (K_G,\Sigma,Q,LS_{\mathcal{FOL}},\delta,q_0,Z_0,F)$ such that:

- $\Sigma = \emptyset$,
- $K_G = \emptyset$,
- $Q = \{q_0,q_1\}$,
- $F = \{q_1\}$,
- $Z_0 = \lambda$,
- $LS_{\mathcal{FOL}} = \left\{\begin{array}{l} AskApply_G(w,r), Apply_G(w,r,neww), Accept_G(w) \mid \\ w \in (N\cup T)^* \wedge r \in P \end{array}\right\}$

If we consider $X \in LS_{\mathcal{FOL}} \cup \{\lambda\}$ as the memory, $w \in (N\cup T)^*$, $A,B,C \in N$ and $a \in T$, then the preconditions of the locutions in $LS_{\mathcal{FOL}}$ are defined bellow:

$prec(AskApply_G(w,r)) = w \in (N\cup T)^* \wedge r \in P$,
$prec(Apply_G(w,r,neww)) = w \Rightarrow_r neww$,
$prec(Acceept_G(w)) = w \in T^*$

- $\delta(q_0, AskApply_G(S,r), \lambda) = \{(q_0, AskApply_G(S,r))\}$,
  $\delta(q_0, AskApply_G(neww, s), Apply_G(w, r, neww)) = \{(q_0, AskApply_G(neww, s))\}$,
  $\delta(q_0, Apply_G(w, r, neww), AskApply_G(w, r)) = \{(q_0, Apply_G(w, r, neww))\}$,
  $\delta(q_0, Accept_G(neww), Apply_G(w, r, neww)) = \{(q_1, Accept_G(neww))\}$.

The following can be proved:

$$\left(\forall w_t \in T^* : \left(\begin{array}{l} \exists\{r_1, ..., r_t\} \in 2^P : S \Rightarrow_G^{r_1} w_1 \Rightarrow_G^{r_2} w_2 ... \Rightarrow_G^{r_{t-1}} w_{t-1} \Rightarrow_G^{r_t} w_t \leftrightarrow \\ \left(\exists \beta \in LS_{\mathcal{FOL}} : \begin{array}{l} (q_0, \beta, \lambda) \Rightarrow_D^* (q_0, Accept_G(w_t), Apply_G(w_{t-1}, r_t)) \Rightarrow_D \\ \Rightarrow_D (q_1, \lambda, Accept_G(w_t)) \end{array} \right) \end{array} \right) \right)$$

$$\leftrightarrow \left\{ \begin{array}{ll} \text{Definition of } D \wedge \beta = & AskApply_G(S, r_1) Appply_G(S, r_1, w_1) ... \\ & AskApply_G(w_{t-1}, r_t) Apply_G(w_{t-1}, r_t, w_t) Accept_G(w_t) \end{array} \right\}$$

true

■

5.4.3 Applicability in Dialogue Theory

Family $lmFST_n$ provides a formal framework to specify dialogue protocols with the following features:

- The number of conversants is n
- The agents cannot learn or modify their own knowledge bases but they can modify the shared knowledge base through the dialogue.
- The memory is restricted to at most a locution.
- Backtracking is not possible.
- Turn-taking and arbitrary replying policies can be used.
- Any locution set is allowed.
- The complexity of agent reasoning strategies is limited to computable functions.
- Arbitrary dialogue initiatives are possible.

5.5 $rCFST$ systems

In this section we introduce a type of $ConvFST$ systems that we call *regularly Controlled Finite State Transition (rCFST)* systems where we consider the following:

- We restrict the shared knowledge to a string of nonterminals and terminal symbols. The nonterminals symbols are from a finite alphabet N, and the terminal symbols are from a finite alphabet T, such that $N \cap T = \emptyset$.
- We restrict the agent private knowledge to a finite set of pairs whose first component is a locution without parameters, and the second component is a non-erasing CF rule.

- We consider only locutions without parameters. For each locution there is a non-erasing CF rule associated and it is decidable to check if the locution's precondition evaluates true. For the precondition of a locution to be satisfied it has to be possible to apply its associated CF rule over the string of shared knowledge. In case a locution can be uttered the shared knowledge is modified applying the CF rule associated with the locution.

5.5.1 Formal Definition

Below we formally define *regularly Controlled Finite State Transition (rCFST)* systems with degree $n \geq 1$:

Definition 61

$$rCFST_n = \left\{ \begin{array}{l} W \in ConvFST_n \mid W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{FOL}}, \Gamma, \delta, q_0, SK_0, F) \wedge \\ LS_{\mathcal{FOL}} = \{\rho_{id_i} \mid 1 \leq i \leq n\} \wedge \\ \left(\forall \rho_{id_i} \in LS_{\mathcal{FOL}} : prec(\rho_{id_i}) = (\exists (\rho_{id_i}, A \rightarrow x) \in K_{id_i}) \wedge (\exists \alpha, \beta : SK = \alpha A \beta) \right) \wedge \\ \Gamma = N \cup T \wedge \ N \cap T \cap \{(,), \rightarrow\} = \emptyset \wedge SK_0 \in N \wedge \Sigma = \Gamma \cup LS_{\mathcal{FOL}} \cup \{(,), \rightarrow\} \wedge \\ \left(\begin{array}{l} \forall 1 \leq i \leq n : K_{id_i} \textit{ is a finite set} \wedge \\ K_{id_i} = \{(\rho_{id_i}, A \rightarrow x) \mid \rho_{id_i} \in LS_{\mathcal{FOL}} \wedge A \in N \wedge x \in (N \cup V)^+\} \end{array} \right) \wedge \\ \left(\begin{array}{l} \forall q, k \in Q, \ newSK, SK \in \Gamma^*, \ \rho_{id_i}\alpha \in (LS_{\mathcal{FOL}})^+ : \\ \left(\begin{array}{l} \exists (\rho_{id_i}, A \rightarrow x) \in K_{id_i} : \\ prec(\rho_{id_i}) \wedge SK \Rightarrow^{A \rightarrow x} newSK \end{array} \right) \rightarrow (q, \rho_{id_i}\alpha, SK) \Rightarrow_W (k, \alpha, newSK) \end{array} \right) \end{array} \right\}$$

We denote by $rCFST_n$, $n \geq 1$, the family of languages $L_{SK}(W)$ where $W \in rCFST_n$. When the number of agents is not bounded, then we replace the corresponding parameter with ∞.

When we are interested only in strings over some alphabet T, hence in the language $L_{SK}(W) \cap T^*$, then we speak about *Extended regularly Controlled Finite State Transition (ErCFST)* systems. We denote by $ErCFST_n$, the family of languages $L_{SK}(W) \cap T^*$ where $W \in rCFST_n$.

5.5.2 An example of dialogue protocol

rCFST systems allow to specify mixed-initiative frame-based dialogue protocols. The shared string $SK \in (N \cup T)^*$ of nonterminals and terminals is used as a frame to guide the dialogue. Such that each nonterminal from N represents a gap that is considered filled when it is rewritten by a terminal symbol from T. The speaker associates questions to gaps in the frame. Then a speaker asks the other speaker questions in order to fill in the gaps in the frame. A speaker can answer a question (rewrite a nonterminal with a terminal) or reply to a question with other question (rewrite a nonterminal with other nonterminal), what is considered a change in the dialogue initiative. When all the gaps in a frame are filled, or equivalently SK is a string of terminals, the dialogue is finished.

In [AJR01] it was exemplified the use of *CD Grammar systems with memories* in the simulation of a 2-party dialogue protocol in a film shop. In their example the customer C tries to buy a film and the seller S tries to find out the film that the customer needs in order to sell it. Basically their dialogue protocol can be seen as a frame-based mixed-initiative protocol, therefore it can be specified by a *rCFST* system. We show it below.

We define $P \in rCFST_2$ such that $P = (K_c, K_s, \Sigma, Q, LS_{\mathcal{FOL}}, N \cup T, \delta, q_0, S, F)$ where:

- $LS_{\mathcal{FOL}} = \{need_c, ask_s, answer_c, answer_s, agree_c, refuse_c\}$.
- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$.
- $F = \{q_4, q_5\}$.
- $N = \{S, F, W, L, A, I, O, ?_P, ?_A, [,]\}$.
- $T = \{Yes, No, a_{100}, ..., a_{1000}, l_8, l_{12}, ..., l_{48}, Kodak, Minolta, Fuji\}$.
- Below we provide some examples of pairs that can be considered as part of K_c:
  - $(need_c, S \rightarrow FW)$ to indicate that the customer goal is to buy a film $F \in N$. The symbol $W \in N$ is introduced to be replaced by a terminal symbol Yes in case the dialogue finishes with the customer buying the film or by terminal symbol No otherwise.
  - $(answer_c, L \rightarrow l_{36}), (answer_c, L \rightarrow l_{24})$ to answer that he needs a film of 36 pictures or 24 pictures, where $L \in N$ represents a question of how many pictures the film needs to have.
  - $(answer_c, A \rightarrow ?_A), (answer_c, A \rightarrow a_{100}), ..., (Answer_c, A \rightarrow a_{1000})$. Nonterminal $A \in N$ represents the question of the speed of the film, then the customer can ask for a clarification of what is speed ($?_A \in N$) or tell the speed ($a_i \in T$).
  - $(answer_c, ?_P \rightarrow I), (answer_c, ?_P \rightarrow O)$. The symbol $?_P \in N$ represents the question of where is the customer going to take the photo, whose answer can be indoors ($I \in N$) or outdoors ($O \in N$).
  - $(agree_c, W \rightarrow Yes)$ to indicate that the customer agrees to buy the film, where $W \in N$ and $Yes \in T$.
  - $(refuse_c, W \rightarrow No)$ to indicate that the customer refuses to buy the film, where $W \in N$ and $No \in T$.
- Below we provide some examples of pairs that can be considered as part of K_c:
  - $(ask_c, F \rightarrow [B][A]L)$ to ask the customer for the length of the film in number of pictures ($L \in N$). Future questions will be the brand and speed in ASA ($[B], [A] \in N$).
  - $(ask_c, [A] \rightarrow A)$ to ask the customer for the speed in ASA ($A \in N$).
  - $(ask_c, ?_A \rightarrow ?_P)$ to reply to the question of 'what is speed?' ($?_A \in N$) with the new question of 'where do you expect to take the photos?' ($?_P \in N$).
  - $(answer_c, [B] \rightarrow Kodak)$ to suggest the brand Kodak with $Kodak \in T$.
  - $(answer_c, O \rightarrow a_{100})$ to suggest that a good speed for photos taken outside is 100 ASA ($a_{100} \in T$).
- Function δ is defined in the following way where $SK \in (N \cup T)^*$ is the shared knowledge:

  $\delta(q_0, need_c, SK) = \{(q_1, newSK)\}$,
  $\delta(q_1, ask_s, SK) = \{(q_2, newSK)\}$,

$\delta(q_2, answer_c, SK) = \{(q_1, newSK)\}$,
$\delta(q_1, answer_s, SK) = \{(q_3, newSK)\}$,
$\delta(q_3, answer_s, SK) = \{(q_3, newSK)\}$,
$\delta(q_3, agree_c, SK) = \{(q_4, newSK)\}$,
$\delta(q_3, refuse_c, SK) = \{(q_5, newSK)\}$.

Speaker	Utterance
C:	I need a film.
S:	Which kind of film?
C:	One for 36 pictures.
S:	Which speed?
C:	Speed? What do you mean?
S:	Where will you take your pictures?
C:	Outside.
S:	I suggest 100 ASA.
S:	I suggest the Kodak brand.
C:	OK.

Figure 5.3: A dialogue instance between a customer C and a seller S

In the dialogue protocol $P \in rCFST_2$ the dialogue instance from Figure 5.3 can be simulated by the following derivation:

$(q_0, need_c(ask_s answer_c)^3 answer_c^2 agree_c, S) \Rightarrow_P$
$(q_1, (ask_s answer_c)^3 answer_c^2 agree_c, FW) \Rightarrow_P$
$(q_2, answer_c(ask_s answer_c)^2 answer_c^2 agree_c, [B][A]LW) \Rightarrow_P$
$(q_1, (ask_s answer_c)^2 answer_c^2 agree_c, [B][A]l_{36}W) \Rightarrow_P$
$(q_2, answer_c ask_s answer_c answer_c^2 agree_c, [B]Al_{36}W) \Rightarrow_P$
$(q_1, ask_s answer_c answer_c^2 agree_c, [B]?_A l_{36}QW) \Rightarrow_P$
$(q_2, answer_c answer_c^2 agree_c, [B]?_P l_{36}W) \Rightarrow_P$
$(q_1, answer_c^2 agree_c, [B]Ol_{36}W) \Rightarrow_P$
$(q_3, answer_c agree_c, [B]a_{100}l_{36}W) \Rightarrow_P$
$(q_3, agree_c, Kodak\ a_{100}l_{36}W) \Rightarrow_P$
$(q_4, \lambda, Kodak\ a_{100}l_{36}Yes)$ with $Kodak\ a_{100}l_{36}Yes \in T^+$.

In figure 5.4 we present the transition graph for the locution interchange in dialogue protocol $P \in rCFST_2$.

5.5.3 Formal Properties

Below we prove that the family $ErCFST_n$ has the same expressive power as the family rC of classes of languages generated by *regularly Controlled* Grammar systems with non erasing CF rewriting rules introduced in [GS68] [DPS97]. From this result we deduce that $ErCFST$ is a type of mildly CS grammar.

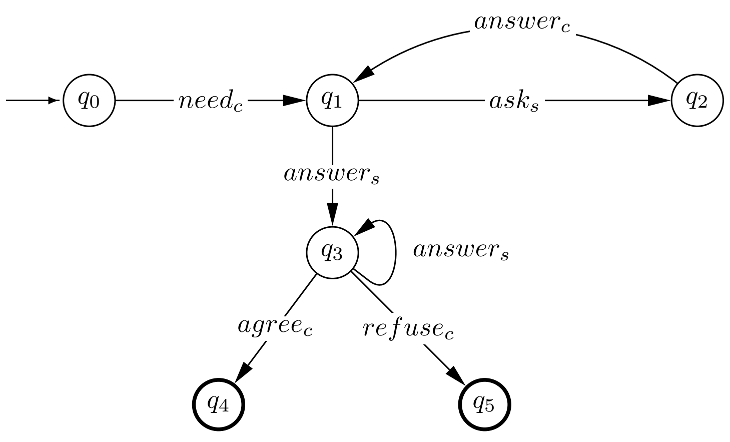

Figure 5.4: Transition graph for the interchange of locutions in protocol $P \in rCFST_2$ between a seller and a customer in a shop

Theorem 22 *$ErCFST_n = rC \subset CS$, for all $n \geq 1$.*

Proof. In [DPS97] it was proved that $rC \subset CS$.

First we prove that $ErCFST_n \subseteq rC$. Given an arbitrary dialogue protocol $W \in ErCFST_n$ such that $W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{FOL}}, N \cup T, \delta, q_0, SK_0, F)$. From W we define $G_W \in rC$, $G_W = (N, T, P, SK_0, R)$ such that:

- $P = \{p_{\rho_{id_i}} \mid 1 \leq i \leq n \wedge p_{\rho_{id_i}} = A \rightarrow x \wedge (\rho_{id_i}, A \rightarrow x) \in K_{id_i}\}$, and
- R is the regular set defined by the finite state automata $AF = (q_0, P, Q, \delta', F)$ where $\left(\begin{array}{l} \forall SK, newSK \in (N \cup T)^*,\ k, q \in Q,\ \rho_{id_i} \in LS_{\mathcal{FOL}} : \\ \qquad\qquad (k, newSK) \in \delta(q, \rho_{id_i}, SK)) \rightarrow k \in \delta'(q, p_{\rho_{id_i}}) \end{array}\right).$

Given the way we defined G_W it is clear that the following is satisfied:

$$\left(\begin{array}{l} \forall m \geq 0, SK_m \in (N \cup T)^*,\ k \in F,\ \alpha \in LS_{\mathcal{FOL}} : \\ (\alpha = \rho_{a_1}...\rho_{a_m} \wedge (q_0, \alpha, SK_0) \Rightarrow^m_W (k, \lambda, SK_m) \leftrightarrow \\ \qquad\qquad (SK_0 \Rightarrow^{p_{\rho_{a_1}}}_{G_W} SK_1 ... \Rightarrow^{p_{\rho_{am}}}_{G_W} SK_m \wedge p_{\rho_{a_1}}...p_{\rho_{a_m}} \in L(R)) \end{array}\right)$$

Then $L_{SK}(W) \cap T^* = L(G_W)$.

Now we prove that $rC \subseteq ErCFST_1$.

Given an arbitrary *regularly Controlled* (context free) grammar $G \in rC$ such that $G = (N, T, P, S, R)$. From G we define the dialogue protocol $W_G \in ErCFST_n$ such that

$$W_G = (K_{id_1}, \Sigma, Q, LS_{\mathcal{FOL}}, N \cup T, \delta', q_0, S, F) \text{ with:}$$

- $K_{id_1} = \{(p_{id_1}, p) \mid p \in P\}$.
- $LS_{\mathcal{FOL}} = \{p_{id_1} \mid p \in P\}$.

- If $AF = (q_0, P, Q, \delta, F)$ is the finite automata generated by regular expression R, then from AF we get the following definition of function δ':
$$\left(\begin{array}{l} \forall k, q \in Q,\ newSK, SK \in (N \cup T)^*,\ p \in P,\ p_{id_{id_1}} \in LS_{\mathcal{FOL}} : \\ \qquad (k, newSK) \in \delta(q, p_{id_1}, SK) \rightarrow k \in \delta'(q, p) \wedge SK \Rightarrow_G^p newSK \end{array} \right)$$

By the way we defined W_G it is clear that the following is satisfied:
$$\left(\begin{array}{l} \forall m \geq 0, SK_m \in (N \cup T)^*,\ \alpha \in L(R),\ k \in F : \\ (\alpha = p_1...p_m \wedge SK_0 \Rightarrow_G^{p_1} SK_1 ... \Rightarrow_G^{p_m} SK_m) \leftrightarrow (q_0, p_{1id_1}...p_{mid_1}, SK_0) \Rightarrow_{W_G}^m (k, \lambda, SK_m) \end{array} \right).$$

Then $L(G) = L_{SK}(W_G) \cap T^*$. ■

In [DPS97] it was proved that in the class rC the membership problem is decidable, the emptiness problem is NP-hard and the finiteness problem is NP-hard. From that result from class rC and from theorem 22 we get the corollary below:

Corollary 7 *In family $ErCFST_n$, $n \geq 1$, the membership problem is decidable, the emptiness problem is NP-hard and the finiteness problem is NP-hard.*

From theorems 13 and 22 we get the following corollary:

Corollary 8 *For all $n, m \geq 1$, regular set X over m populations in $ErCREG_m(X)$:*

$$ErCFST_n = ErCREG_m(X)$$

With theorem 23 we provide a restriction under which the family $rCFST_n$ of classes of languages collapses into the family $rCFST_1$.

Theorem 23
$$\left(\begin{array}{l} \forall W \in rCFST_n : W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{FOL}}, N \cup T, \delta, q_0, SK_0, F) \wedge \\ K_{id_{n+1}} = K_{id_1} \cup ... \cup K_{id_n} \wedge \\ \left(\exists W' \in rCFST_1 : RenameL(L_{Dg}(W), id_{n+1}) = L_{Dg}(W') \wedge L_{SK}(W) = L_{SK}(W') \right) \end{array} \right),$$
for all $n \geq 1$.

Proof.

Considering an arbitrary $W \in rCFST_n$ such that $W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{FOL}}, \Gamma, \delta, q_0, SK_0, F)$. From W we can define $W' \in rCFST_1$ such that $W' = (K_{id_{n+1}}, \Sigma, Q, newLS_{\mathcal{FOL}}, \Gamma, \delta', q_0, SK_0, F)$ where:

- $K_{id_{n+1}} = \{(\rho_{id_{n+1}}, A \rightarrow x) \mid (\rho_{id_i}, A \rightarrow x) \in K_{id_i} \wedge 1 \leq i \leq n\}$,
- $newLS_{\mathcal{FOL}} = \{\rho_{id_{n+1}} \mid \rho_{id_i} \in LS_{\mathcal{FOL}} \wedge 1 \leq i \leq n\}$, and
- $$\left(\begin{array}{l} \forall p, q \in Q,\ \rho_{id_i} \in LS_{\mathcal{FOL}},\ SK, newSK \in (LS_{\mathcal{FOL}})^*) : \\ \qquad (p, newSK) \in \delta(q, \rho_{id_i}, SK) \rightarrow (p, newSK) \in \delta'(q, \rho_{id_{n+1}}, SK) \end{array} \right).$$

The proof is similar to the proof of theorem 15, except that here we do not require the following hypothesis:

$(\forall \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \rightarrow prec(\rho_{id_{n+1}}(v^{(h)}))) \wedge$
$\Big(\forall \rho_{id_{n+1}}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_{n+1}}(v^{(h)})) \rightarrow \Big(\exists \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \Big) \Big)$

The reason why we do not need it is that if we take an arbitrary locution $\rho_{id_u}(v^{(k)}) \in LS_{\mathcal{FOL}}$, $1 \leq u \leq n$, the following can be proved:

$prec(\rho_{id_u}(v^{(k)}))$
$\leftrightarrow$ {Definition of $ErCFST$ systems and definition of $RenameLc$}
$(\exists(\rho_{id_u}, A \rightarrow x) \in K_{id_u} : RenameLc(\rho_{id_u}(v^{(k)}), id_{n+1}) = \rho_{id_{n+1}}(v^{(k)}))$
$\leftrightarrow$ { Definition of W' and definition of $RenameLc$}
$(\exists(\rho_{id_{n+1}}, A \rightarrow x) \in K_{id_{n+1}})$
$\leftrightarrow$ {Definition of $ErCFST$ systems}
$prec(\rho_{id_{n+1}}(v^{(k)}))$. ■

5.5.4 Applicability in Dialogue Theory

Class $rCFST_n$ provides a formal framework to specify dialogue protocols with the following features:

- The number of conversants is n
- The agents cannot learn or modify their own knowledge bases but they can modify the shared knowledge base through the dialogue.
- Backtracking, turn-taking and arbitrary replying policies and systems initiatives can be used.
- The locution set is limited to locutions without parameters.
- An agent private knowledge base is restricted to a finite set of pairs whose first component is a locution without parameters and the second component is a CF rule.
- The agent's decision procedures are fixed and they consist on checking if there is a pair in the agent private knowledge base whose first component is the locution that the agent wants to utter and the second component is a CF that the agent can apply over the string corresponding to the shared knowledge.

Chapter 6

Conclusions and Future Work

Dialogue Theory is a very old subject with roots in Aristotle's days. But it is in the 70's that this research area re-appeared in analytical Philosophy as a structure to evaluate argumentation and informal fallacies [MH69] [Pol74]. Dialogue Theory has recently being regarded as an important subject of research with multiple applications and potential uses that scientists from areas like Formal Language Theory, Artificial Intelligence, Logic, Linguistics and Argumentation Theory can fruitfully explore.

The aim of our thesis was to contribute to Dialogue Theory providing a hierarchy of formal frameworks for the study of dialogues. For developing this hierarchy we mainly applied notions from Formal Language Theory, and Grammar System Theory in particular.

Below we summarize our contributions and proposals of future work according to the scientific area to which our results specifically apply to:

6.1 Contributions and future work in Dialogue Theory

A popular way to express the social norms between groups of agents is by means of an explicit *protocol*. Dialogue protocols specify complex concurrent and asynchronous patterns of communication between agents in a society and they are used to state their social norms. A society protocol is examined in-advance by an agent in order to decide if it joins or not the corresponding society. The protocol also acts as a guide for the agents once they are operating within the society. Dialogue protocols correspond to dialogue state theory. In Dialogue Theory protocols have been defined by means of finite automata [Vas04, ERS$^+$01], high level Petri Nets [HK98][MW97][PC96], different diagrams provided by the Unified Modeling Language [Woo99] [OPB00] [WCW01] [HWW01], logic [Woo99] [GK94] and process descriptions [Rob04] [Wal04c].

Based on our knowledge of Formal Language Theory we contributed to Dialogue Theory introducing a hierarchy of formal frameworks for the definition of dialogue protocols as transition systems.

In chapters 4 and 5 we defined the frameworks that we proposed for specifying dialogue protocols. In chapter 4 we presented the frameworks based on Grammar system variants and in chapter 5 the frameworks based on finite state transition systems. Each framework defines a generic, abstract and well defined formalism in which protocols can be specified, compared and evaluated in the same notation in a precise way. For the frameworks we defined we explored some of their formal properties: expressive power, computational complexity, decidability of the problem of determining if a dialogue instance satisfies a dialogue protocol (membership problem), etc. For the frameworks that we introduced we analyzed their expressive power in terms of the Chomsky hierarchy and in terms of features like: the number of speakers, or the way the speakers perceive the context (in block, scattered) or the length of the context they can perceive, etc. Besides we mentioned the practical applicability of the frameworks we proposed for the specification of dialogues and we compared them using less formal properties taken from the Theory of Dialogues, like: backtracking, turn-taking, dynamic change of agent's knowledge bases and shared knowledge bases, restrictions on the maximum number of speakers, flexibility in agent reasoning and learning strategies, flexibility in dialogue initiative, etc.

In chapter 4 we defined *Conversational Enlarged Reproductive Eco-Grammar* (*ConvEREG*) systems. This variant of Grammar systems proved to be adequate for modeling goal-oriented dialogue protocols. A preliminary definition of *ConvEREG* has been published in [BEGJL06] [EGJL06] [EGJL07]. *ConvEREG* systems are the result of the combination of different approaches undertaken in Dialogue Theory: the MAP^a language with application in multi-agent systems area (Artificial Intelligence), Conversational Grammar systems (Formal Language Theory) and studies of human dialogues (Linguistics). *ConvEREG* systems are based on a Grammar system variant that we introduced with the name of *Enlarged Reproductive Eco-Grammar (EREG)* system, where the grammatical components are interpreted as speakers whose participation in the conversation is described by protocols and whose decisions are taken by invocating computable functions. The protocols we considered are strings in a process calculus that we defined as the *Multi Agent Protocol (MAP^a)* language [GW06b] [GW06c] [GW06a]. In previous grammatically based formal models for the specification of dialogue protocols the participation of the speakers in the conversation was given by rewriting rules. While strings can be changed during run-time according to the dialogic state, rewriting rules are defined at design time and they remain fixed during all the conversation. Since the *ConvEREG* systems do not have this restriction they can be used to specified conversations with dialogic spaces that can not be known beforehand, but that emerge through the dialogue. Besides *ConvEREG* systems allow the specification of protocols where the focus is on the agents' observable behavior abstracting from the agent implementation details. Therefore this framework can be used to specify dialogues involving agents that are not fully knowable or whose complexity of representation is high. Another features characteristic of the protocols specified within *ConvEREG* systems are: they abstract from natural language processing issues, they correspond to information state theory, they are linguistically well founded and they can exhibit a high level of generality and modularity. In chapter 4 we explained those features in detail and we analyzed why *ConvEREG* systems can be considered an improvement with respect to previous approaches for the specification of dialogue protocols based on Grammar systems [CVJLMV99] [JL00] [AJR01].

Also in chapter 4 we introduced a type of *ConvEREG* system that we called *Conditional Eco-Grammar* (*CondEG*) systems. *CondEG* systems are a variant of Eco-Grammar systems where each agent's private knowledge base is restricted to two finite sets of conditional rules. In each derivation step every agent inspects its two sets of conditional rules in order to decide its participation in the conversation and how to change its mental state. The first set of rules is used by the agent to decide how to change the dialogue context (environment) according to the presence and\or absence of information in its mental state (agent state). The second set of rules is used by the agent to change its mental state according to the presence and\or absence of information in the dialogue context. The decision processes undertaken by the agents are given by computable functions. *ECondEG* systems restricted to non erasing rules are a subclass of CS grammars.

We completed chapter 4 presenting another type of *ConvEREG* system that we called *regularly Controlled Reproductive Eco-Grammar (rCEREG)* system. *rCEREG* systems are variants of Reproductive Eco-Grammar systems to which we attached a turn talking policy given by a regular set which determines the order that speakers must respect to participate in the dialogue. Because in *ErCERED* systems we restricted agents to invocate only computable functions it turned out that this framework has the same expressive power as *regularly Controlled (rC)* grammars with non-erasing CF rules [GS68][DPS97]. The family *rC* corresponds to a type of mildly CS grammar where the membership problem is decidable, the emptiness problem is NP-hard and the finiteness problem is NP-hard.

In chapter 4 we analyzed the influence that the number of speakers, the length of the permitting, the length of forbidding contexts and the type of conditions (scattered or blocking) considered have over the expressive power of the *ECondEG* and *ErCREG* systems:

1. $ECondEG_n(i, j, c) \subseteq ECondEG_m(i', j', c)$, for all $c \in \{b, s\}$, $1 \leq n \leq m$ and $i \leq i', j \leq j'$.
2. $ECondEG_n(i, j, b) \subseteq ECondEG_n(i, j, s)$, for all $n \geq 1$ and $i, j \in \{0, 1\}$.
3. $ECondEG_n(0, 0, c) = ET0L$, for all $n \geq 1$, $c \in \{b, s\}$.
4. $ECTEG_n(i, j, c) \subseteq ECondEG_1(i, j, c)$, for all $i, j \geq 0$, $n \geq 1$, $c \in \{b, s\}$.
5. $ECondEG_{e,n}(i, j, c) \subseteq CS$, for all $n \geq 1$, $i, j \geq 0$, $c \in \{b, s\}$.
6. $ErCREG_n(X) = rC \subset CS$, for all $n \geq 1$, regular set X over n populations.

We presented in chapter 5 the frameworks based on finite state transition systems that we defined for specifying dialogues. First we explained a framework that we called *Conversational Finite State Transition (ConvFST)* system. We presented a preliminary version of this approach in [GP07] [Gra08]. *ConvFST* systems allow to specify dialogue protocols where the number of agents is fixed, agents are provided with fix sets of private beliefs, the interaction model is described by a finite state transition system whose transitions are conditionals and labeled with locutions, and the shared knowledge is saved in a string whose content can be modified. The states of the automata correspond to possible stages of the conversation and the transitions represent dialogue moves. For a labeled transition to be triggered the precondition associated with the locution that labels it has to be satisfied. The locution's precondition is a formula over some logic which is evaluated according to the agent knowledge and shared knowledge. The evaluation is given by a computable function. If a transition is triggered a new state is reached and some operation can be performed over the shared knowledge. *ConvFST* systems are a type of dialogue state approach. We showed the applicability of this framework for the definition of the argumentation-based dialogue protocols introduced in [Amg98]. Besides we provided an example of the possible benefits of specifying a dialogue in the framework *ConvFST*, where properties can be proved. We proved that *EConvFST* have the same expressive power as Turing Machines.

We defined *Conditional Finite State Transition (CondFST)* systems as a type of *ConvFST* systems where the agents are provided with a finite set of conditional rules that they use to decide their participation in the dialogue. The agents use these rules to decide how to modify the shared knowledge depending on the presence and\or absence of substrings(information) in that string. The decision processes undertaken by the agents are given by computable functions. When the language of share knowledge generated is considered *EConvFST* systems with non erasing rewriting rules are a subclass of CS grammars.

We introduced *limited memory Finite State Transition (lmFST)* systems as a type of *ConvFST* systems. While in *ConvFST* systems the shared knowledge is given by a string of symbols from a finite alphabet, in *lmFST* systems the shared knowledge is restricted to at most the last uttered locution. Because there are no restrictions over the locutions used very complex dialogues can be modeled, where locutions can contain information of previously uttered locutions and even the whole dialogue history. When the language of share knowledge generated is considered *ElmFST* systems have the same expressive power of CS grammars. We investigated for the case of the argumentation-based dialogues from [Amg98] the practical implicants of restricting the memory to a locution.

We finished chapter 5 specifying *regularly Controlled Finite State Transition (rCFST)* systems as a type of *ConvFST* systems. In *rCFST* systems the locutions have no parameters and each locution is associated with a CF rule such that the locution's precondition consists on checking if the associated CF rule can be applied over the string of shared knowledge. If the precondition associated with a locution is satisfied, then the locution is uttered and a new string of shared knowledge is obtained from the application of the CF rewriting rule associated with that locution. We investigated the suitability of *rCFST* systems for specifying frame-based mixed-initiative dialogues. When in *ErCFST* systems the language of share knowledge generated is considered, this framework has the same expressive power as the *regularly Controlled (rC) grammars* with non-erasing CF rules.

The frameworks presented in chapter 5 can be seen as a continuation of similar formal models based on finite-state automata introduced in [Vas04, ERS+01] [HK98][MW97][PC96] for the simulation of dialogues.

In chapter 5 we proved some results for *ConvFST*, *CondFST*, *lmFST* and *rCFST* systems. Bellow we list these results were we analyzed the influence that the number of speakers, the length of the permitting and forbidding contexts and the type of conditions (scattered or blocking) considered can have over the expressive power of the families presented in chapter 5:

1.
$$\left(\begin{array}{l} \forall W \in ConvFST_n : W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{L}}, \Gamma, \delta, q_0, SK_0, F) \wedge \\ K_{id_{n+1}} = K_{id_1} \cup ... \cup K_{id_n} \wedge \\ \left(\begin{array}{l} \forall \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \rightarrow prec(\rho_{id_{n+1}}(v^{(h)})) \wedge \\ \left(\forall \rho_{id_{n+1}}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_{n+1}}(v^{(h)})) \rightarrow \left(\exists \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \right) \right) \rightarrow \\ \qquad\qquad \left(\begin{array}{l} \exists W' \in ConvFST_1 : RenameL(L_{Dg}(W), id_{n+1}) = L_{Dg}(W') \wedge \\ L_{SK}(W) = L_{SK}(W') \end{array}\right) \end{array}\right) \end{array}\right),$$
for all $n \geq 1$.

2. $ECondFST_n(i, j, c) \subseteq ECondFST_m(i', j', c)$, for all $c \in \{b, s\}$, $1 \leq n \leq m$ and $i \leq i'$, $j \leq j'$.

3. $ECondFST_n(i, j, b) \subseteq ECondFST_n(i, j, s)$, for all $n \geq 1$ and $i, j \in \{0, 1\}$.

4. $ECondFST_n(0, 0, c) = ETOL$, $n \geq 1$, $c \in \{b, s\}$.

5. $$\left(\begin{array}{l} \forall W \in CondFST_n : W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{L}}, \Gamma, \delta, q_0, SK_0, F) \wedge \\ K_{id_{n+1}} = K_{id_1} \cup ... \cup K_{id_n} \wedge \\ \left(\exists W' \in CondFST_1 : RenameL(L_{Dg}(W), id_{n+1}) = L_{Dg}(W') \wedge L_{SK}(W) = L_{SK}(W') \right) \end{array} \right),$$
for all $n \geq 1$.

6. $ECTEG_n(i, j, c) \subseteq ECondFST_1(\infty, \infty, c)$, for all $n \geq 1$, $i, j \geq 0$, $c \in \{b, s\}$

7. $ECondFST_{e,n}(i, j, c) \subseteq CS$, for all $n \geq 1$, $i, j \geq 0$, $c \in \{b, s\}$.

8. $$\left(\begin{array}{l} \forall W \in lmFST_n : W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{L}}, \Gamma, \delta, q_0, Z_0, F) \wedge K_{id_{n+1}} = K_{id_1} \cup ... \cup K_{id_n} \wedge \\ \left(\begin{array}{l} (\forall \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \to prec(\rho_{id_{n+1}}(v^{(h)}))) \wedge \\ \left(\forall \rho_{id_{n+1}}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_{n+1}}(v^{(h)})) \to \left(\exists \rho_{id_i}(v^{(h)}) \in LS_{\mathcal{L}} : prec(\rho_{id_i}(v^{(h)})) \right) \right) \to \\ \quad \left(\exists W' \in lmFST_1 : RenameL(L_{Dg}(W), id_{n+1}) = L_{Dg}(W') \wedge L_{SK}(W) = L_{SK}(W') \right) \end{array} \right) \end{array} \right),$$
for all $n \geq 1$.

9. $ElmFST_n = CS$, for all $n \geq 1$.

10. $ErCFST_n = rC \subset CS$, for all $n \geq 1$.

11. $$\left(\begin{array}{l} \forall W \in rCFST_n : W = (K_{id_1}, ..., K_{id_n}, \Sigma, Q, LS_{\mathcal{FOL}}, N \cup T, \delta, q_0, SK_0, F) \wedge \\ K_{id_{n+1}} = K_{id_1} \cup ... \cup K_{id_n} \wedge \\ \left(\exists W' \in rCFST_1 : RenameL(L_{Dg}(W), id_{n+1}) = L_{Dg}(W') \wedge L_{SK}(W) = L_{SK}(W') \right) \end{array} \right),$$
for all $n \geq 1$.

In figure 6.1 we present the hierarchy that connects the frameworks that we defined in chapters 4 and 5 with the Chomsky hierarchy: if two families are connected by a line (an arrow), then the upper family includes properly (includes) the lower family; if two families are not connected then they are not necessarily incomparable.

With the hierarchy that we introduced and the results we proved we contributed to Dialogue Theory with a formal and comparative study of formal frameworks for the specification of dialogue protocols defined as transition systems. This type of comparative study is frequent in Formal Language Theory, while in the area of multi-agent systems the formal frameworks introduced for specifying communication are usually partially compared by studying the satisfaction of sets of desirable dialogue features. The research we presented here can help designers of protocols in the area of Artificial Intelligence, and multi-agent systems in particular, for the task of selecting the most suitable formalism for the definition of dialogues that fulfil better their requirements of expressible power and computational complexity.

We are interested on continuing the study of the hierarchy we proposed, trying to prove for the existing frameworks more properties and analyzing the introduction of new frameworks inspired in practical requirements from the field of Artificial Intelligence. It is our intention to continue with this research in an interdisciplinary way, trying to provide formal solutions from Formal Language Theory to practical problems introduced in the area of Dialogue Theory.

An open problem in the area of Dialogue Theory is the absence of a formal strategy to compare arbitrary dialogue protocols. In a future we are interested on studying this issue to address the type of problems introduced in [PWA03b]:

- How might one choose between two protocols?, when is one protocol preferable to another?
- When do two protocols differ?, can we tell if a protocol is new (in the sense of providing a different functionality from an existing protocol rather than just having equivalent locution with different names)?;
- Is a protocol new (in the sense of providing a different functionality from an existing protocol rather than just having equivalent locutions with different names)?.

In [GGF08] we worked on the comparison of concurrent processes defined in the framework of Coloured Petri Nets, focusing only on the labeled transition systems generated and abstracting from the agent's reasonings strategies. We based our study on the notions of simulation equivalence defined in the process algebra called pi-calculus [Mil99]. Examples of simulation equivalences that we considered are branching time similarity, observable similarity, strong similarity, week similarity, etc. Our intention was to recognize processes that were bisimilar to be able to interchange them. But as the example 3 shows, a deeper comparison of dialogues where the rationality of the agents is considered should be studied from formal frameworks as the ones we defined on this work.

6.2 Contributions and future work in Grammar systems

In Formal Language Theory rewriting rules are precise and unambiguous ways to describe local changes. But getting an intuition of the global state of a grammar from its set of rewriting rules is undecidable and harder when the complexity of the grammar definition increases. In the case of Grammar system the emergent, parallel and distributed behavior increases even more the computational complexity of the model and the level of difficulty in the proofs. Until now mainly analysis by cases has been the formal strategy used for proving properties in Grammar system.

Although the aim of our thesis was to explore possible ways to apply Grammar System Theory to the formal study of dialogues, in chapter 3 *we analyzed a way to contribute to Grammar System Theory with knowledge from Multiprogramming area, addressing the problem of formally proving that a Grammar system verifies certain properties or that it produces a desired outcome.*

Because the programming field is more developed in the area of formal strategies of proof we studied the possibility of incorporating strategies and results from that area. Our proposal, presented in [GM04a] [GM04b] and [GM07], consisted on interpreting Grammar systems as programs in order to benefit from strategies for derivation, reasoning and proving from Multiprogramming framework. The analogy between programs as transformers of predicates and Grammar systems as derivation systems is clear: the same as each statement in a program modifies the state of the system, every rewriting rule in the Grammar system modifies the string(s) under derivation.

Following this approach we obtained the following results:

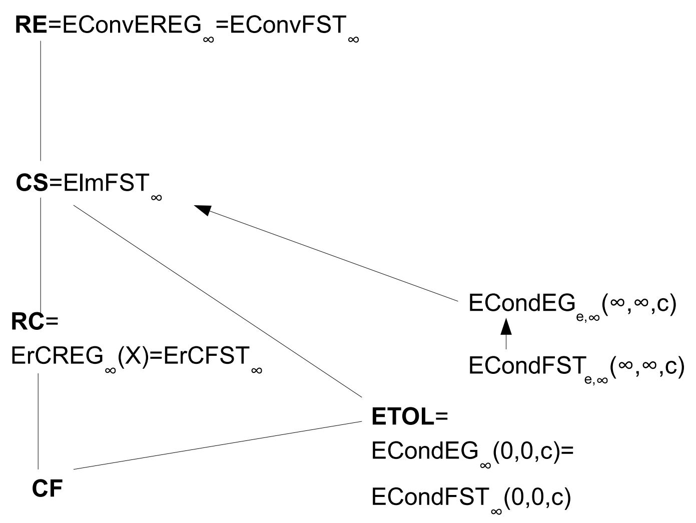

Figure 6.1: Hierarchy of frameworks defined

- We proved that an arbitrary CD Grammar systems, PC Grammar systems (centralized, non centralized, returning and non returning), CCPC Grammar systems or Eco-Grammar systems can be interpreted as an imperative concurrent programs where each grammar is a program running concurrently

- We provided an alternative approach to solve the problem of *given a Grammar system prove that it generates a specific language*. Instead of using the strategies available in Grammar systems, mainly analysis by cases, we proposed to translate the Grammar system into a multiprogram and solve it using Owicki-Gries Theory and some strategies developed in the programming framework for the development of programs, like for instance *problem refinement* or *functional decomposition*.

- We introduced a new approach for solving the problem of *given a language specification find a Grammar system that generates the given language*. So far this problem has been solved using the following strategy: first propose a Grammar system and then prove by means of language theory that the proposed Grammar system generates indeed the given language. We exemplified how Owicki-Gries logic of programming can guide us in obtaining *simultaneously* a Grammar system that generates the given language and the proof that it generates it. This new approach might be of a great benefit for the Grammar Systems Theory where there is no formal strategy of this type. We showed how to apply this strategy for a well-known non-context free language, namely $L_{cd} = \{a^n b^m c^n d^m \mid n, m \geq 1\}$. In [Chi97] it was proved that the cross agreement language L_{cd} can be generated by a returning non-centralized PC Grammar system with three context free components ($L_{cd} \in PC_3(CF)$). We improved this result showing that L_{cd} can be generated by a non-returning non-centralized PC Grammar system with three right regular components ($L_{cd} \in NPC_3(Reg)$)[GM07]. We provided thus a solution to an open problem mentioned in [Chi97]. This can be considered an improvement because the complexity of the grammatical components is reduced from context free grammars to regular grammars.

- We argued that Owicki-Gries methodology provides a strategy of proof close-to-human way of reasoning:

  - It allows to reason in a forward or data-driven way, similarly to analysis by cases technique, but also in a backward or goal-directed way. The notion of backward reasoning comes from psychology, as it is pointed out in [KC58] where this description of problem solving occurs:

    "We may have a choice between starting with where we wish to end, or starting with where we are at the moment. In the first instance we start by analyzing the goal. We ask, "Suppose we did achieve the goal, how would things be different- what subproblems would we have solved, etc.?". This in turn would determine the sequence of problems, and we would work back to the beginning. In the second instance we start by analyzing the present situation, see the implications of the given conditions and lay-out, and attack the various subproblems in a *forward direction*."

  - It provides strategies to reason about the problems in terms of subproblems, what according to [KC58] reflects the way humans think:

    " The person perceives in his surrounding goals capable of removing his needs and fulfilling his desires [...] And there is the important phenomenon of emergence of subgoals. The pathways to goals are often perceived as organized into a number of subparts, each of which constitutes and intermediate subgoal to be attained on the way to the ultimate goal."

- We showed that for some problems it is possible to use a strategy of proof from Owicki-Gries Theory that is called *system invariant*. Applying this strategy requires to find a predicate that remains invariant through all the computation and that synthesizes the behavior of the multiprogram. The main advantages of the *system invariant strategy* over the *analysis by cases technique* are:

  - The number of proofs reduce to linear size.
  - It captures the global behavior of the system by an invariant that shows all possible values of the sentential form and hides information that the analysis by cases gives. So we can say that invariant system captures global behavior in a *more abstract way*.

- We argued that interpreting Grammar systems as multiprograms we can improve their time efficient applying available programming techniques, like *functional decomposition*, for increasing the level of parallelism.

We are interested on continuing our research on the field of formal strategies of proof for Grammar systems, trying to find more connections between the areas of Grammar System Theory and the Multiprogramming area.

In chapter 3 we explained that Reproductive Eco-Grammar systems can not be translated to the DGC language because this language does not provide commands to dynamically create new agent instances during run-time. We would like to study the translation of Reproductive Eco-Grammar systems to the Object-Oriented (OO) paradigm in Computer Science, where objects (agent instances) can be dynamically created. Once a Reproductive Eco-Grammar systems is translated to an OO language two options are possible:

1. Try some Hoare-based formal verification strategies available, like for instance the ones explained in [Rey82], [AL97], [dB99], [PdB03], [PHM99], [RWH01] and [vO01].

2. Try some co-algebraic strategies of proof proposed for object-oriented languages. For example in [Jac96] and [Rei95] they introduce verification systems where classes in an OO language are described as co-algebras, which may occur as models (implementations) of co-algebraic specifications. An object belonging to a class is an element of the state space of the class, as co-algebra.

Reproductive Eco-Grammar systems have proved to be adequate (able to capture many real life-like features) and theoretically relevant (they allow inference of nontrivial conclusions). If we can also provide them with formal proof strategies we will increase their practical relevance as formal frameworks for the study and analysis of problems in multiple areas like Ecology, Economy, Social systems, Robotic systems, etc.

6.3 Future work in Concurrent Programming

We proved that $L_{cd} \notin X_2(Reg)$, for $X \in \{PC, CPC, NPC, NCPC\}$, or equivalently that our result $L_{cd} \in NPC_3(Reg)$ is optimal. This kind of results provide evidence that in Grammar System Theory analysis by cases can be used for proving negative results of the type: a given language cannot be generated by any Grammar system of a specified type. So far the formal strategies available in the programming field are: *verification* and *formal derivation*. Both strategies are useful to get positive results, but so far no formal strategies has been introduced in the programming framework to cope with the problem of finding whether a given output cannot be generated by any multiprogram behaving in a certain way. We consider that this can be an interesting topic of research to be addressed in the future.

In Grammar System Theory there are some theorems that allow to transform a Grammar system of m grammars in a Grammar system of n grammars that generate the same language. For instance considering $CD_{n,*}$ the family of CD grammar systems with at least n grammars, the following results are proved in [CVJJP94]:

$$CF = CD_{1,*}(t) = CD_{2,*}(t) \subset CD_{3,*}(t) \text{ and} \tag{6.1}$$

$$CD_{3,*}(t) = CD_{*,*}(t) = ET0L \tag{6.2}$$

The theorems mentioned above can be considered a contribution from Grammar System Theory to the programming framework, where there are no results concerning the number of programs needed to solve a problem. It would be very interesting for the design of concurrent programs if some of these transformations can also consider efficiency. If there would be some results concerning with how to transform a program $\mathcal{P}$ that has m multiprograms running concurrently into a program with n multiprograms that solve the same problem more efficiently, this could be a great contribution to Multiprogramming Theory.

Therefore we consider that a possible future direction of research can be to find ways in which Multiprogramming field can benefit from the formal research in Formal Language Theory.

Summarizing our thesis focused on the formal study of dialogue interaction. Our approach can be seen as an interdisciplinary research in the area of Dialogue Theory where some theories and results from Formal Language Theory and another areas of study like Artificial Intelligence, Linguistics or Multiprogramming were analyzed and connected.

Bibliography

[ABP06] L. Amgoud, S. Belabbes, and H. Prade. A formal general setting for dialogue protocols. In *Artificial Intelligence: Methodology, Systems, and Applications, 12th International Conference, AIMSA 2006, Varna, Bulgaria, September 12-15, 2006, Proceedings*, volume 4183 of *Lecture Notes in Computer Science*, pages 13–23. Springer, 2006.

[AFS01] J. F. Allen, G. Ferguson, and A. Stent. An architecture for more realistic conversational systems. In *Intelligent User Interfaces*, pages 1–8, 2001.

[AJR01] S. Aydin, H. Jürgensens, and L. Robbins. Dialogue as co-operating grammars. *Journal of Automata, Languages and Combinatories*, 6:395–410, 2001.

[AL97] M. Abadi and R. Leino. A logic of object-oriented programs. In *TPSOFT 97: Theory and Practice of Software Development, 7th International Joint Conference CAAP/FASE*, volume Lecture Notes in Computer Science 1214, pages 682–696. Springer-Berlag New York, 1997.

[Amg98] L. Amgoud. On the aceptability of arguments in preference-based argumentation framework. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 1–7, 1998.

[B9̈6] Th. Bäck. *Evolutionary Algorithms in Theory and Practice - Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, 1996.

[BCEL99] P. Bohlin, R. Cooper, E. Engdahl, and S. Larsson. Information states and dialogue move engines. In Jan Alexandersson, editor, *Proceedings of the IJCAI-99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pages 25–31, Murray Hill, New Jersey, 1999. International Joint Conference on Artificial Intelligence.

[BD93] A. Burns and G. Davies. *Concurrent Programming.* Addison-Wesley, 1993.

[BEGJL06] G. Bel-Enguix, M. A. Grando, and M. D. Jimenez-Lopez. Distributed protocols and distributed grammars: Two theories for agent communication. In M. P. Huget, R. van Eijk, and R. A. Flores, editors, *AAMAS Workshop on Agent Communication (AC 2006)*, pages 35–49, 2006.

[BS96] P. Breiter and M.D. Sadek. A rational agent as a kernel of a cooperative dialogue system: Implementing a logical theory of interaction. In *Proceedings ECAI-96 Workshop Agent Theories, Architectures, and Languages*, pages 261–276. Springer-Verlag, Berlin, 1996.

[CA00] B. Cases and M. Alfonseca. Eco-grammar systems applied to the analysis of the dynamics of population in artificial worlds. In R. Freund and A. Kelemenová, editors, *Proceeding of the International Workshop Grammar Systems 2000*, pages 265–281. Silesian University, 2000.

[CC00] J. Chu-Carroll. Mimic: An adaptive mixed initiative spoken dialogue system for information queries. In *Proceedings ANLP 6*, pages 97–104, 2000.

[Chi97] A. Chitu. PC grammar systems versus some non-context free constructions from natural and artifical languages. *New Trends in Formal Languages, LNCS*, 1218:278–287, 1997.

[Cla96] H. H. Clark. *Using Language*. Cambridge University Press, 1996.

[CP79] P. R. Cohen and C. R. Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3(3):177–212, 1979.

[CVD90] E. Cshuhaj-Varjú and J. Dassow. On cooperating/distributed grammar systems. *Journal of Information Processing and Cybernetics EIK*, 26, (1-2):49–63, 1990.

[CVJJP94] E. Csuhaj-Varjú, J.Dassow, J.Kelemen, and Gh. Păun. *Grammar Systems: A Grammatical approach to distribution and cooperation*. Gordon and Breach Science Publishers, 1994.

[CVJKP94] E. Csuhaj-Varjú, A. Kelemenová J. Kelemen, and Gh.Gh. Păun. Eco (grammar) systems: a preview. *Cybernetics and Systems'94*, pages 941–948, 1994.

[CVJL98] E. Csuhaj-Varjú and M. D. Jiménez-López. Cultural eco-grammar systems - a multi-agent system for cultural change. In A. Kelemenová, editor, *Proceedings of the MFCS'98 Satellite Workshop on Grammar Systems*, pages 165–182. Silesian University, 1998.

[CVJLMV99] E. Csuhaj-Varjú, M. D. Jiménez-López, and C. Martín-Vide. Pragmatics and eco-rewriting systems. In Gh. Păun and A. Salomaa, editors, *Grammatical Models of Multi-Agent Systems*, pages 262–283. Gordon and Breach, 1999.

[CVKKP97] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, and Gh. Păun. Eco-grammar systems: a grammatical framework for studying lifelike interactions. *Artifitial Life*, 3(1):1–28, 1997.

[CVKP96] E. Csuhaj-Varjú, J. Kelemen, and Gh. Păun. Grammar systems with wave-like communication. *Computers and Artificial Intelligence*, 15(5), 1996.

[CVPS95] E. Csuhaj-Varju, G. Paun, and A. Salomaa. Conditional tabled eco-grammar systems. *Journal of Universal Computer Science*, 1(5):252–268, 1995.

[Das95] J. Dassow. An example of an eco-grammar system: a can collecting robot. In Gh. Paun, editor, *Proceeding of the Workshop on Artificial Life*, pages 240–244. The Black Sea University Press, 1995.

[dB99] F. S. de Boer. A wp-calculus for OO. In *Proceedings of the Second International Conference on Foundations of Software Science and Computation Structures*, volume 1578, pages 135–149. Lecture Notes in Computer Science, 1999.

[DE76] E. Dijkstra and W. Edsger. *A Discipline of Programming*. Prentice-Hall Series in Automatic Computation, 1976.

[DG05a] A. H. Dediu and M. A. Grando. Eco-grammar systems as models for parallel evolutionary algorithms. In A. V. Chaskin O. B. Lupanov, O. M. Kasim-Zade and K. Steinhöfel, editors, *Proceedings of Stochastic Algorithms: Foundations and Applications, Third International Symposium (SAGA 2005)*, volume 3777, pages 228–238. Springer, 2005.

[DG05b] A. H. Dediu and M. A. Grando. Simulating evolutionary algorithms with eco-grammar systems. In J. Mira and J. R. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, volume 3562(2), pages 112–121. Springer, 2005.

[DP97] J. Dassow and Gh. Păun. *Handbook of Formal Languages*, volume 2, chapter Grammar Systems, pages 155–214. Springer-Verlag, 1997.

[DPS97] J. Dassow, Gh. Păun, and A. Salomma. *Handbook of Formal Languages*, volume 2, chapter Grammars with Controlled Derivations, pages 101–154. Springer-Verlag, 1997.

[Dre92] H. Dreyfus. *What computers still can't do*. The MIT Press, 1992.

[DS90] E. Dijkstra and C. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag, 1990.

[Dun95] P. M. Dung. On the aceptability of arguments and its fundamental role in non-monotonic reasoning logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.

[EBHM03] R. M. Van Eijk, F. S. De Boer, W. Van Der Hoek, and J. Ch. Meyer. A verification framework for agent communication. *Autonomous Agents and Multi-Agent Systems*, 6(2):185–219, 2003.

[EGJL06] G. Bel Enguix, M. A. Grando, and M. D. Jiménez-López. A grammatical framework for modelling multi-agent dialogues. In *9th Pacific Rim International Workshop on Multi-Agents (PRIMA 2006)*. Springer-Verlag, 2006.

[EGJL07] G. Bel Enguix, M. A. Grando, and M. D. Jiménez-López. An interaction protocol for agent communication. In *Software Agents and Services for business Research and E-sciences (SABRE 2007)*. Springer Berlin Heidelberg, 2007.

[Err93] L. Errico. WAVE: An overview of the model and the language. *CSRG, Dept. Electronic and Electr. Eng., University of Surrey, UK*, 1993.

[ERS$^+$01] M. Esteva, J. A. Rodriguez, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specification of electronic institutions. *Lecture Notes in Artificial Intelligence, Agent-mediated Electronic Commerce (The European AgentLink Perspective)*, pages 126–147, 2001.

[Fos95] I. Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley, 1995.

[GGF08] M. Adela Grando, D. W. Glasspol, and J. Fox. Petri nets as a formalism for comparing expressiveness of workfow-based clinical guideline languages. In *6th Internalcional Conf. on Business Process Management (BPM 2008), ProHealth Workshop, Milan, Italy*, volume 17, pages 348–360. Lecture Notes in Bussiness Information Processing, Springer, 2008.

[GK94] M. R. Genereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 1994.

[GK97] T. F. Gordon and N. Karacapilidis. The zeno argumentation framework. In *Proceedings of the Sixth International Conference on AI and Law*, pages 10–18. ACM Press, 1997.

[GM04a] A. Grando and V. Mitrana. Can PC grammar systems benefit from concurrent programming? In E. Csuhaj-Varjú and G. Vaszil, editors, *Preproceeding of Grammar System Week 2004*, pages 179–187. MTA SZTAKI, 2004.

[GM04b] A. Grando and V. Mitrana. A possible connection between two theories: Grammar systems and concurrent programming. In E. Csuhaj-Varjú and G. Vaszil, editors, *Proceedings of Grammar System Week 2004*, pages 200–211. MTA SZTAKI, 2004.

[GM07] M. A. Grando and V. Mitrana. A possible connection between two theories: Grammar systems and concurrent programming. *Fundamenta Informaticae, Special issue on developments in grammar systems*, 76(3):325–336, 2007.

[Gor94] T. F. Gordon. The pleadings game: An exercise in computational dialectics. *Artificial Intelligence and Law*, (2):239–292, 1994.

[GP07] M. A. Grando and S. Parsons. Dialogues as extended finite transition systems. In G. Bel Enguix and M. D. Jiménez-López, editors, *Internacional workshop on nonclassical formal languages in linguistics (ForLing) 2007*, 2007.

[Gra08] M. A. Grando. Simulating dialogues with finite state transition systems. In *Proceeding of 8 Congreso de Linguistica General*, 2008.

[GS68] S. Ginsburg and E. H. Spanier. Control sets on grammars. *Mathematical Systems Theory*, 2(2):159–177, 1968.

[GW06a] A. Grando and C. Walton. MAP^a: a language for modelling conversations in agent environments. In *Intelligent Information Processing and Web Mining Conference 2006 (IIPWM06)*, volume 35. Springer Verlag, 2006.

[GW06b] A. Grando and C. Walton. The MAP^a language of agent dialogues. In *Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS06)*, pages 1375–1377. ACM Press, New York, 2006.

[GW06c] A. Grando and C. Walton. Specifying protocols for knowledge transfer and action restriction in multiagent systems. In *10th International Workshop on Cooperative Information Agents (CIA2006)*, volume 4149, pages 11–13. Springer-Verlag, Berling, 2006.

[HK98] T. Holvoet and T. Kielmann. Behaviour specification of parallel active objects. *Journal on Parallel Computing*, 24(7):1107–1135, 1998.

[HKK94] J. Hromkovič, J. V. Kari, and L. V. Kari. Some hierarchies for the communication complexity measures of cooperating grammar systems. *Theoretical Computer Science*, 127(1):123–147, 1994.

[Hoa69] C. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[HWW01] J. L. Koningand M. P. Huget, J. Wei, and X. Wang. Extended modeling languages for interaction protocol design. In *Proceedings of Agent-Oriented Software Engineering (AOSE-01)*, 2001.

[Jac96] B. P. F. Jacobs. Objects and classes, coalgebraically. In B. Freitag, C. B. Jones, C. Lengauer, and H. J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer Academic Publishers, Boston, 1996.

[JL99] M. D. Jiménez-López. *Cultural Eco-Grammar Systems: Agents between Choice and Imposition. A Preview*, pages 181–187. Springer Hungarica, 1999.

[JL00] M. D. Jiménez-López. *Grammar Systems: a formal-language-theoretic gramework for linguistics and cultural evolution*. PhD thesis, Rovira i Virgili University, 2000.

[JL01] M. D. Jiménez-López. *Linguistic Grammar Systems: A Grammar Systems Approach for Natural Language*, pages 55–66. Gordon and Breach, 2001.

[KC58] D. Krech and R. S. Crutchfield. *Elements of Psychology*. Alfred A. Knopf, 1958.

[KNS93] S. Kraus, M. Nirkhe, and K. P. Sycara. Reaching agreements through argumentation: a logical model (preliminary report). In *Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, pages 233–247, Hidden Valley, Pennsylvania, 1993.

[Lar05] S. Larsson. Dialogue systems: Simulations or interfaces? In *Proceedings of the Ninth Workshop on the Semantics and Pragmatics of dialogue (DIALOR'05)*, pages 45–52, 2005.

[LRG06] N. Ludwig, P. Reiss, and G. Görz. Conald: The configurable plan-based dialogue system. In *2006 IAR Annual Meeting*. Deutsch-Französisches Institut für Automation und Robotik, 2006.

[LT00] S. Larsson and D. Traum. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering*, 6(3–4):323–340, 2000.

[MC81] J. Misra and K.M. Chandy. Proofs of networks of processes. *IEEE Trans. Software Eng.*, 7(4), 1981.

[McB02] P. McBurney. *Rational Interaction.* PhD thesis, University of Liverpool, 2002.

[ME98] N. Maudet and F. Evrard. A generic framework for dialogue game implementation. In N. Maudet and F. Evrard, editors, *Proceedings of the Second Workshop on Formal Semantics and Pragmatics of Dialog*, Universite Twente, The Netherlands, 1998.

[MH69] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[Mih94] V. Mihalache. Extended conditional tabled eco-grammar systems. *Journal of Information Processing and Cybernetics*, 30(4):213–229, 1994.

[Mih95] V. Mihalache. General artificial intelligence systems as eco-grammar systems. In Gh. Păun, editor, *Proceeding of the Workshop on Artificial Life*, pages 245–259. The Black Sea University Press, 1995.

[Mih99a] V. Mihalache. Decidability problems in grammar systems. *Theoretical Computer Science*, 215, 1999.

[Mih99b] V. Mihalache. On the expressiveness of coverability trees for PC grammar systems. In Gh. Păun and A. Salomaa, editors, *Grammatical Models of Multi-Agent Systems*, pages 86–98. Gordon and Breach, 1999.

[Mil99] R. Milner. Communicating and mobile systems: the pi-calculus. *Cambridge Univ. Press*, 1999.

[MVM00] C. Martin-Vide and V. Mitrana. Parallel communicating automata systems, a survey. *Korean J. Comput. Appl. Math.*, pages 237–257, 2000.

[MW97] D. Moldt and F. Wienberg. Multi-agent-systems based on coloured petri nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets (ICATPN '97)*, number 1248, pages 82–101. Lecture Notes in Computer Science, 1997.

[OG76] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs 1. *Acta Informatica*, 6:319–340, 1976.

[OPB00] J. Odell, H. V.D. Parunak, and B. Bauer. Extending UML for agents. In *Proceedings of Agent-Oriented Information Systems Workshop (AOIS-00)*, 2000.

[Par93] D. Pardubská. On the power of communication structure for distributive generation of languages. *Developments in Language Theory, At the Crossroads of Mathematics, Computer Science and Biology*, pages 419–429, 1993.

[PC96] M. Purvis and S. Craneffeld. Agent modelling with petri nets. In *Proceedings of the CESA '96 (Computational Engineering in Systems Applications) Symposium on Discrete Events and Manufacturing Systems*, pages 602–607, 1996.

[PdB03] C. Pierik and F. S. de Boer. A syntax-directed Hoare logic for Object-Oriented programming concepts. Technical report UU-CS-2003-010, Institute of Information and Computing Sciencs, Utrecht University, 2003.

[PFPS$^+$92] R. Patil, R. Fikes, P. Patel-Schneider, D. McKay, T. Finin, T. Gruber, and R. Neches. The DARPA knowledge sharing effort: Progress report. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, 1992.

[PHM99] A. Poetzsch-Heffter and P. Müller. A programming logic for sequential Java. In S. D. Swierstra, editor, *ESOP 99*, volume 1576, pages 162–176. Lecture Note in Computer Science, 1999.

[PMW04] S. Parsons, P. McBurney, and M. Wooldridge. The mechanics of some formal inter-agent dialogues. In *Lecture Notes in Computer Science: Advances in Agent Communication*. Springer Berlin / Heidelberg, 2004.

[Pol74] J. Pollock. *Knowledge and Justification*. Princeton University Press, 1974.

[PS89] Gh. Păun and L. Sântean. Parallel communicating grammar systems: the regular case. *Ser. Matem.-Inform 38*, pages 55–63, 1989.

[PV00] H. Prakken and G. Vreeswijk. Logical systems for defeasible argumentation. In Kluwer Academic Pub, editor, *Handbook of philosophical logic*, 2000.

[PWA02] S. Parsons, M. Wooldridge, and L. Amgoud. An analysis of formal inter-agent dialogues. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 394 – 401. ACM Press, 2002.

[PWA03a] S. Parsons, M. Wooldridge, and L. Amgoud. On the outcomes of formal inter-agent dialogues. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 616–623. ACM Press, 2003.

[PWA03b] S. Parsons, M. Wooldridge, and L. Amgoud. Properties and complexity of some formal inter-agent dialogues. *Journal of Logic and Computation*, 13(3):347–376, 2003.

[Ree98] C. Reed. Dialogue frames in agent communication. In *Proceedings of the 3rd International Conference on Multi Agent Systems (ICMAS98)*, pages 246–253, France, 1998.

[Rei95] H. Reichel. An approach to object semantics based on terminal co-algebras. *Mathematical Structures in Computer Science*, 5(2):129–152, 1995.

[Rey82] J. C. Reynolds. Idealized ALGOL and its specification logic. *Tools and Notions for Program Construction*, pages 121–161, 1982.

[Rob04] D. Robertson. A lightweight coordination calculus for agent systems. In Springer Berling/ Heidelberg, editor, *Declarative Agent Languages and Technologies II*, volume Lecture Notes in Computer Science 3476/2005, pages 183–197, 2004.

[RS97] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*. Springer-Verlag, 1997.

[RWH01] B. Reus, M. Wirsing, and R. Hennicker. A Hoare calculus for verifying Java realizations of OCL-constrained design models. In *FASE 2001*, volume 2029, pages 300–317. Lecture Notes in Computer Science, 2001.

[SM94] Gh. Stefan and M. Malita. The eco-chip: A physical support for artificial life systems. In Gh. Păun, editor, *Artificial Life: Grammatical Models. Proceedings of the Workshop on Artificial Life*, pages 273–288. The Black Sea University Press, 1994.

[Sos96] P. Sosík. On eco-grammar systems and artificial neural networks. *Computers and Artificial Intelligence*, 15:247–264, 1996.

[SP94] M. Srinivas and L. M. Patnoik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24 (4):656–668, 1994.

[SP00] S. Seneff and J. Polifroni. Dialogue management in the Mercury flight reservation system. In *ANLP/NAACL 2000 Workshop on Conversational systems*, pages 11–16, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[SPR98] M. Schroeder, D. A. Plewe, and A. Raab. ULTIMA RATIO: Should Hamlet kill Claudius? In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 467–468, New York, 9–13, 1998. ACM Press.

[SS04] P. Sebestyén and P. Sosik. Multiple robots in space: An adaptive ecogrammar model. In *Preproceeding of Grammar System Week 2004*, pages 284–298, 2004.

[SSJ74] H. Sacks, E. A. Schegloff, and G. Jefferson. A simplest systematics for the organization of turn-taking for conversation. *Language*, 50(4):696–735, 1974.

[Syc89] K. Sycara. Argumentation: Planning other agents' plans. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989.

[TE93] F. L. Tiplea and C. Ene. A coverability structure for parallel communicating grammar systems. *Journal of Information Processing and Cybernetics*, EIK 29 (5):303–315, 1993.

[TEIP94] F. L. Tiplea, C. Ene, C. M. Ionescu, and O. Procopiuc. Some decision problems for parallel communicating grammar systems. *Theoretical Computer Science*, 134 (2):365–385, 1994.

[TKI97] F. L. Tiplea, M. Katsura, and M. Ito. Processes and vectorial characterizations of parallel communicating grammar systems. *Journal of Automata, Languages and Combinatorics*, 2(1):47–78, 1997.

[TL03] D. Traum and S. Larsson. The information state approach to dialogue management. *Current and New Directions in Discourse and Dialogue*, pages 325–353, 2003.

[TP05] Y. Tang and S. Parsons. Argumentation-based dialogues for deliberation. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 552–559, New York, NY, USA, 2005. ACM Press.

[Vas04] W. Vasconcelos. Norm verification and analysis of electronic institutions. In *2004 Workshop on Declarative Agent Languages and Technologies (DALT-04)*, pages 141–155, 2004.

[vEGH$^+$96a] F. H. van Eemeren, R. Grootendorst, F. S. Henkemans, J. Anthony Blair, R. H. Johnson, E. C. W. Krabbe, C. Plantin, D. N. Walton, C. A. Willard, J. Woods, and D. Zarefsky. *Fundamentasl of Argumentation Theory: A Handbook of Historical Backgrounds and Contemporary Developments*. LAWRENCE ERLBAUM ASSOCIATES, 1996.

[vEGH96b] F. H. van Eemeren, R. F. Grootendorst, and F. S. Henkemans. *Fundamentals of Argumentation Theory: A Handbook of Historical Backgrounds and Contemporary Applications*. Lawrence Erlbaum Associates, Hillsdale NJ, USA, 1996.

[vO01] D. von Oheimb. Hoare logic for Java in Isabelle/HOL. *Concurrency and Computation: Practice and Experience*, 13:1173–1214, 2001.

[VO02] M. Viroli and A. Omicini. Specifying agent observable behaviour. In *1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pages 712–720. ACM, 2002.

[Wal04a] C. Walton. Model checking agent dialogues. In *Proceedings of the 2004 Workshop on Declarative Agent Languages and Technologies (DALT)*, volume Lecture Notes in Computer Science 3476/2005, pages 132–147. Springer Berlin / Heidelberg, 2004.

[Wal04b] C. Walton. Model checking multi-agent web services. In *Proceedings of the 2004 Spring Symposium on Semantic Web Services*, 2004.

[Wal04c] C. Walton. Multi-agent dialogue protocols. In *Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics*, 2004.

[WCW01] J. Wei, S. C. Cheung, and X. Wang. Towards a methodology for formal design and analysis of agent interaction protocols: An investigation in electronic commerce. In *Proceedings of the International Software Engineering Symposium*, 2001.

[Wei66] J. Weizenbaum. ELIZA a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[WGS87] J. Widom, D. Gries, and F. B. Schneider. Completeness and incompleteness of trace-based network proof systems. In *POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 27–38, New York, NY, USA, 1987. ACM Press.

[WK95] D. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY press, 1995.

[Woo67] W. A. Woods. Semantics for a question-answering system. Technical Report NSF-19, Harvard University Computation Laboratory, 1967.

[Woo73] W. A. Woods. Progress in natural language understanding: an application to LUNAR geology. In *Proceedings of AFIPS Natl. Comput. Conf. Expo.*, number 42, pages 441–450, 1973.

[Woo99] W. A. Woods. FIPA specification: agent communication language. Technical Report Part 2, Foundation for Intelligent Physical Agents, 1999. Available at: www.fipa.org.

[Woo00] M. Wooldridge. Semantic issues in the verification of agent communication languages. *Autonomous Agents and Multi-Agent Systems*, 3(1):9–31, 2000.