Università degli Studi di Roma Tre
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

An Algebraic Semantics for the LO Coordination Language

Giulio Balestreri[1]

RT-DIA-32-98 **Febbraio 1998**

(1) Università di Roma Tre,
Via della Vasca Navale, 79
00146 Roma, Italy

ABSTRACT

In this paper we investigate the relation between polynomial reduction and the coordination model LO. In particular, we give an algebraic semantics of such language mapping objects into 0. The practical outcome that we expect from this theoretical effort is the definition of a framework for statically analyzing and reasoning about LO and Gamma programs. We define an evolving state as a set of agents shared by some external application and observe that such objects can be described as a reduction ring [22, 55]. These rings enjoy interesting rewriting properties and allow us to develop a semantic interpretations in it.

The aim of this approach is to characterize the computations of an *open system* by the use of a method taking in considerations the natural symbolic features representing the living agents. This approach could help in reasoning about the different kinds of agent evolutions such as termination, creation and transformation, in terms of computational algebra.

1 Introduction

The technology of languages for distributed systems programming is continuously improving and one of the main problems arising with the development and reuse of such systems is due to the vast amount of information that has to be managed and analyzed. In fact, in order to have a complete view of the work that a "large" system is really performing, it is necessary to take into account many different facets of the system: problems related to concurrency and communication, peculiarities due to the kind of communication, whether it is synchronous or asynchronous, sequential execution of tasks and also the particular goals of individual agents.

In the classical approaches to concurrent programming [34, 28, 53, 48, 42, 33, 44], all these issues are addressed simultaneously and treated at the same level, without making any explicit distinction between problems related to communication and computation problems. At present, in fact, many concurrent applications use a set of *ad hoc* patterns for the coordination of the different active components, loosing the possibility of describing a communication protocol consisting of a collection of primitives concerning solely communication tasks.

In other words, programmers have to interact directly with low level built-in functions in order to carry out a concurrent application. The use of such primitives is mostly scattered around in the source code and it is mixed up with those parts of code that are not concerned with communication. It results, therefore, that the program does not emphasize the cooperation aspect in a sensible way, in that no piece of code dealing exclusively with such an aspect can be easily identified. As a consequence, the part of code ruling communication and coordination cannot be designed, developed, debugged and maintained separately, nor can it be reused for other applications.

In the last years, parallel to the development of distributed systems constituted by autonomous and possibly eterogenous agents, a new class of models, formalisms and mechanisms for describing concurrent and distributed computations has emerged. These models for "coordinated" computing tend to take apart explicitly the two aspects of concurrent computing: computations and communication [25]. The description of the structure of a system is distinguished from the algorithmic part describing the behavior of individual processes. In this view, the communication issue is considered at a different level with respect to "pure" computations. Every computation model can, in principle, be equipped with coordination functions and, on the other hand, a same set of coordination primitives can be used with different sets of agents to perform different tasks.

In [35] a parallel has been drawn with the situation at the birth of the logic programming paradigm. Kowalski's equation

$$\text{Algorithm} = \text{Logic} + \text{Control}$$

has been rephrased into:

$$\text{Concurrency} = \text{Computation} + \text{Coordination}$$

The explicit separation between communication and computations is not only an advantage for the clean treatment of concurrency, but also and mainly makes it easier the reuse of large systems. For example, once a given result has been obtained with a given system configuration, it may be enough to change the coordination code in order to obtain a completely different performance.

The design and implementation of coordination languages follow naturally the sep-

aration of the communication management task in concurrency models. Their kernel is centered on the aspects related to communication. As it often happens when more importance is recognized to some concept, it has been given a new name: *"coordination"*.

The evolution of concurrent programming to a sort of cooperative computation becomes a "must" when considering the interaction among eterogenous components, possibly relying on different software and hardware, whose features are moreover subject to change. The design of coordination languages focuses on the tools and mechanisms that allow the realization of such a powerful agent as an *orchestra director* ([6]), having a view of the global behavior of the system and able to manage information flow as easily as possible.

Probably one of the best known coordination language is Linda. Its name comes from the cleanliness purposes that inspired its design: coordination and computation are taken apart, and the focus is on the complete specification of a set of built-in functions for the management of information exchange, that can be linked to different programming languages in building an application.

Many issues have to be addressed in the field of coordination languages and, being these formalisms quite recent, there is no universally accepted view of the main problems. For example:

- what is the nature of the coordinated entities, whether they are objects, processes, procedures;
- what is the coordination architecture: message passing models, Petri nets, blackboard models, streams;
- what are the coordination protocols and rules: multi-set rewriting, message passing, or even HTML, as a powerful language for data exchange but with a weak coordination protocol.

In the rest of this section we give an example showing what changes in the passage from a purely concurrent programming style to a coordinated one. The example is from Arbab [14] and it stresses the point in the particular case of his language *MANIFOLD*. It clarifies in general the question in the case of languages based on message passing models, showing the transition from Milner's CCS to a new coordination oriented version. It must be said, however, that MANIFOLD is not the only evolution of CCS towards coordination. Other proposals have been made, stemming from different points of weakness in CCS management of coordination. In [51], for example, the Calculus for Coordinating Environments (CCE) is presented, where CCS syntax is strongly exploited, but with the addition of CCE agents able to coordinate compositions of CCS agents.

Example 1.1 *Let p and q be two processes, that will be treated in the following in a CCS-like syntax, and let us assume that p is to produce some data (two numbers, for example), with which q can compute its result, that must in turn be passed over to p. The pseudo-code for this application is the following:*

process p	*process q*
compute n	*receive n*
send n to q	*let z be sender of n*
compute m	*receive m*
send m to q	*do r = f(n, m)*
do other things	*send r to z*
receive r	
do g(r)	

We note immediately that the code contains both a description of the computations performed by p and q, and a description of how they interact. Moreover, an asymmetry in the communication operations can be observed: in fact, the send operation by p, thus p itself, have to know who is to receive the number, while when the q process receives the message, it needs not know who is the sender. Comparing p and q, it can be noticed that q is more loosely dependent from the environment than p. This means that q is in some sense better reusable than p and that it can furnish its services also to different processes in different applications. On the other hand, if a different application required that the result r of q's computations is not to be sent back to the sender p, but rather to another process t, then again a line of code would have to be changed. However, this would be a change in the way how q, p and t cooperate and not in their own computational tasks; the need to change q's code is due to the presence of the two-argument operation send "what" "to whom".

The MANIFOLD system admits communication ports analogous to CCS's ones, but *anonymous communication* is supported: in general an agent needs not necessarily know the agents he cooperates with and is not responsible of the communication needed to receive or send data. The first of MANIFOLDS's standpoints is in fact that *no process is responsible of its own communication with other processes.* In this view, two kind of agents are distinguished: "workers", whose task is confined to production, and "managers", whose responsibility is the management of the cooperation. Moreover, the manager/worker relation is hierarchical, in the sense that a manager T, coordinating the operations of a set of worker processes, can in turn be seen as a worker by other higher degree manager processes.

The above described concurrent activity is described in MANIFOLD as follows:

process p	process q	process c
compute n	receive n from input port $i1$	create channel ($p.o1$,$q.i1$)
send n to output port $o1$	receive m from input port $i2$	create channel ($p.o2$,$q.i2$)
compute m	do $r = f(n, m)$	create channel ($q.o1$,$p.i1$)
send m to output port $o2$	send r to output port $o1$	
do other things		
receive r from input port $i1$		
do $g(r)$		

The two processes p and q are workers and c is the manager. The workers know nothing about the source of their input and the destination of their output. They can be embedded in different applications and never have to care about the correctness of their input. The coordination model is now explicit and is included only in c's code.

1.1 Shared-memory vs. Channel based Models

A first distinction in coordination languages has followed the two main communication models, based on shared memory and message passing. In the first case, information exchange is managed by means of a set of primitives to access to a shared data repository. Communication is here considered sometimes as a side effect, in that this approach hides the explicit exchange of information. The languages based on the message passing model, on the contrary, have primitives for the explicit exchange of data, but they also need to make synchronization mechanisms explicit in order to coordinate the objects' activities.

An evolution analogous to what the previous example shows, has affected the approaches to concurrent programming based on the shared memory paradigm. The "classical" shared memory communication models assume that each coordinated entity possesses its own code for the management of the interface of the entity itself with the shared memory. In such systems, therefore, the coordination code is embedded in each participant. As a consequence, none of them has a global view of the behaviour of coordination. The main coordination models are based on a common data repository as a communication medium and use some data selection mechanisms as communication primitives. A common feature of such coordination models is that data structures are no more represented as recursively or iteratively defined objects. Rather, they share a topological view of data types, that allows the objects involved in a computation to simultaneously perform several distinct activities. Furthermore no artificious structure is imposed on data, when it is not required.

Coordination languages such as Linda [24, 25], LO [13] and Gamma [17], have multiset as primary data. For instance, Gamma is a formalism manipulating multiset and performing a high level parallel computation. It is based on the idea of describing computations as forms of chemical reactions on a collection of active pieces of data [18]. With this principle, Gamma programs represent the evolution of the data space as a model that is totally without order, where several possibly concurrent steps may occur during the computation.

1.2 Coordination

Coordination is, the process by which an agent reasons about its local actions and the actions of others, and try to ensure that the community (also named global state in the following) acts in a coherent manner. The problems that coordination discusses, are those connected to participation in any computational (or even social) situation. Agents should make contribution to the computational situations providing resources and opportunities, which would otherwise be unavailable. Without coordination the benefits of decentralized problem solving vanishes and the global computation easily collapses generating a chaotic group of single threads. Ensuring that all the single parts of a global task are included in the activities of an agent is one of the objectives of coordination, with the aim of integrating their duties into a overall solution.

For instance, only through coordinated action some solutions can be developed: global constraints, or dependencies between actions are some examples. When an agent starts its life and tries to solve a goal, it may happen that some of its decisions have an impact on the decisions of other members. Another kind of problem that coordination addresses is the one connected to knowledge [7, 8]. When no individual has sufficient competence,

resources or information to solve a problem, or when a goal cannot be solved by individuals because they do not posses sufficient expertise or sufficient strength, coordination (sometimes) brings the necessary help. Classical daily examples are, playing a symphony, driving an airplane or lifting a big and heavy object. Anyway, even when agents can independently work, coordination could be essential: information discovered by one agent can be broadcasted to others, so that, it would be possible to avoid conflicting and redundant effort of the global system and saving computational time.

In order to develop interacting systems, that collaborate in order to solve a common problem, it is important to study models and reasoning techniques that computational agents might share. If we consider a number of agents that work together in a meaningful manner, then first of all, specific description of what an agent is must be given.

2 Agents

Even if there are several and controversial points about what an agent is, there is almost a general agreement on some features and requirements on the nature of agents. In [56] key notions and features of agents are given, both in a computational framework and in a more general sense. An agent, for example, can be decomposed into several independent objects, not physically shared with other individuals. This leads to imagine an agent in much the same way a person is composed of cells living concurrently. Computationally, an agent is a unit of *individual* software, that interfaces with human processes or other agents.

An agent lives in a concurrent and asynchronous computing world like a net; this means that it faces with multiple activities which are distributed and the agents must therefore to communicate. Possibly propagation delay in sending and receiving messages, has to be also considered. The world of collaborative computing is also *open*, meaning that the entities and their environments are always changing. In an open system no overall compile time is present, i.e. when a large system is running, at any time a new agent may be projected and compiled and eventually it may enter in the system. Computation is therefore considered as a simulation of part of the real world, and when the problems become larger and more realistic, they are modeled in terms of concurrent and distributed computing; agents are in fact the actors [2] of such systems.

Another feature of the agents, is the *uncoupling communication* property. This reflects the autonomy and time independence that agents have with respect to each others. Agents are free from following any rule to access data, so that previous models for concurrency, where the communication via shared variables needs explicit declaration of producer consumer relationships [4, 34, 48, 49, 50, 47], are no more suited.

2.1 Interaction

The LO formalism considered in this paper uses a shared data space as the essential means of interaction between objects. In this respect, we focus on systems consisting of agents with multiple goals and interacting among themselves; in this sense coordination languages should be exploited (by agents) both as a ground instance of mutual understanding ([19]) and as a support for share resources and working toward some common goals. Clearly, at a low level, coordination is simply a form of mediating among agents; i.e. a help to

bridge differences among heterogeneus components.

2.2 Reactive vs. Proactive Systems

In order to behave autonomously, agents should possess perception means and the ability to interpret incoming data. Explicit distinctions among agents arise when reasoning, planning, making decisions and executing plans are some of the characteristics of the objects. A typical difference is the one between reactive and intensional agents (see [52]): in the first case an agent is not able to reason about its beliefs. It simply reacts to conditions and performs actions as the result of triggering rules, it has the possibility of updating a private space and sending messages to others, but no problem solving capability is present. On the other hand an intensional agent is able to create plans of actions and to select its goals.

The reactive behavior can also be addressed in terms of rules: well known scripting languages such as Visual Basic [45] or Java Script [37] embody a reactive computational model; using constructs like **on** `mouse-up` **do** `print-string`, with which the reactive behavior of the system is expressed in terms of occurring events that generate new ones. The reactive approach has been successfully implemented in object oriented systems since it works with any kind of *structure*: the objects can be thought of as black boxes that do not care about synchronizations, they are simply coordinated because they *react* to external stimuli.

On the other hand approaches like CLF [10], have a so called *pro active* interpretation of rules: agents' requests are active triggers that request another agent to generate a particular element. For instance, agents should watch for information that meets defined criteria and try to reuse the knowledge they have acquired from executing a certain request in other contexts [8]. As a consequence, a great number of cooperation strategies have been developed, ranging from *master slave* to *sharing common goals* and many others. The protocol adopted by a pro-active vision of the problem must be described in detail for each object. In fact agents should be aware that they are being coordinated and must support a specific code to coordinate and agree with others participants.

3 Linear Objects

Linear Objects (LO) is a language proposed by Andreoli and Pareschi [6, 12] to describe parallel and distributed programs managing autonomous and heterogeneous agents in a distributed environment. It is based on rules with a declarative flavor. Its theoretical background is Linear Logic [1, 31, 32] and it is based on a multiset notion of data. The initial inspiration was given in [5] and it is based on the fact that proof search procedures, are often intrinsically non deterministic. In that paper, it appears that many alternative choices, during a proof expansion, are irrelevant, with the consequence that many inference figures of the sequent system are permutable, while the different orders of their application lead only to a syntactical difference. In [5] a more sophisticated representation of sequents is introduced to the aim of maintaining more information about the status of the proof search. This technique is called *focusing* and lead to refinements strategies which eliminate many irrelevant choices in proof search, keeping the approach sound and complete w.r.t. standard linear logic: any correct proof of linear logic can be mapped into a focusing proof

by simple permutations of the inference figures. LO is strongly based on these ideas and inherits, from linear logic, also Girard's claim *"formulae as resources"* [41, 32]. In fact, its basics objects are multi-sets: bag of elements collected without any relationship among them, neither order nor multiplicity relations. Similarly to Gamma, also in this model the elements can be thought as molecules freely moving imitating the chemical solution behavior: when the molecules come in contact they can react according to some reaction rules. This will be explicitly analyzed in the following, when the Interaction Abstract Machine metaphor is introduced.

Reaction rules are described in terms of LO methods; these similar Prolog clauses, are constituted by conditional rewriting rules on multisets. In its body an LO rule keeps both the reaction condition and the action condition. When a bag satisfies the reaction condition, then it can be rewritten in the way imposed by the action condition.

LO represents and manages coordination introducing the concept of *agent*. An agent is a multiset of *tokens* and a set of rules; each agent can evolve by the use of these rules that it try to apply (*triggering*) to its state (multiset). Communication among agents is expressed in terms of token broadcasting transmission.

Informally we can describe agents' transitions classifying them into three main types: there are the so called *transformations*, in which we have a simple change of the state, *duplications* in which we find a sort of agent cloning or better the creation of a new agent, and finally the *terminations* in which agents disappear and die. Furthermore there is an important operation, which is the broadcasting: it is crucial for communication properties and it is obtained by adding an occurrence of the *broadcasted* token to the state of each agent.

The fact that LO is based on Linear Logic means that the execution of an LO query consists of searching a proof, with a tree like form. Such a search generates a succession of trees represented with their roots at the bottom and growing upwards. In the agent oriented interpretation of such a proof construction, a leaf node corresponds to an agent and a node bottom-up inference evolution corresponds either to a transformation or a cloning or an agent termination. The internal nodes of a proof, can be thought of as the *story* of an agent, while open nodes (leaves) are the *actual state* of the agents, and edges between nodes are old transitions:

$$\textit{system evolution} \leftrightarrow \textit{proof search process}$$

The fundamental idea on which the coordination model LO is based is the need of a layer allowing interactions between coordinated agents, in order to respect their autonomous capability of computing. In fact, if a coordination language contains calls to specific interfaces of an object, it cannot be reused as it is when the object's features change. LO by contrast assumes that the coordination code is external to the individual agents, belonging to a sort of membrane which is the coordinator of the participants. The coordinator undertakes this responsibility, being the unique object that has a global vision of what the global computation should be.

The LO language is based on a set A of atomic formulae, and it consists of two classes of linear formulae, namely "goals" G and "methods" M, recursively defined as follows:

- each atom a is a goal g
- a special goal $\top$ is in G

- if g_1 and g_2 are goals so are $g_1 \& g_2$ and $g_1 @ g_2$
- for $a \in A$ and g goal, $a \circ\!\!- g$ is a method
- if m is a method and $a \in A$ then $a@m$ is a method

Remark Note that @ has higher precedence than $\circ\!\!-$ so it results that for a method $m = a \circ\!\!- b\&c$ and an atom e, $e@m$ is $e@a \circ\!\!- b\&c$.

An LO *program* is a set of methods and a *context* is a set of ground goals. An LO *sequent* is a pair (P, C) (usually written as $P \vdash C$) where P is a program and C is a context. The program evolution is given by a set of Linear logic inference figures. These are:

- decomposition inference figures (applied deterministically to non flat contexts)

$$\frac{P \vdash C, G_1, G_2}{P \vdash C, G_1 @ G_2}\ @ \qquad \frac{P \vdash C, G_1 \quad P \vdash C, G_2}{P \vdash C, G_1 \& G_2}\ \& \qquad \frac{}{P \vdash C, \top}\ \top$$

- progression inference figure (applied deterministically to flat contexts)

$$\frac{P \vdash C, G}{P \vdash C, A_1, ..., A_r} \circ\!\!-$$

where $A_1 @ A_2, ..., @A_r \circ\!\!- G$ is a ground instance of a method of the program P

Example 3.1 *$P = \{a@a@b \circ\!\!- b\&c, g \circ\!\!- h\&(h@k)\}$ is a program with two methods, and $\{a, a, c, b\}$ is a context.*

These inference figures determine a reduction relation between LO-proofs: given two proofs Π_1 and Π_2, Π_1 reduces to Π_2 if Π_2 can be obtained from Π_1 by use of one of the inference figures above. The following definition introduces the concept of **target** proofs (P, g).

Definition 3.1 *A target proof for a sequent $P \vdash G$ is a proof such that its root node is a sequent of the form $P \vdash G, C$ where C is a context containing only atoms.*

Target proofs are searched in such a way that the context at their root node properly contains the initial resource of the program call. In [11] two proof construction mechanisms are considered:

expansion let ν be an open leaf of Π whose sequent matches the lower sequent of an inference figure. Let Π' be a proof obtained by expanding Π at node ν with branch(s) to new open leave(s) labeled with the upper sequent(s) of the selected inference figure. Then Π is expanded to Π' ($\Pi \Rightarrow_e \Pi'$)

insertion let Π' be a proof obtained by adding an occurrence of a given atom to the context at each node of Π, then Π' is obtained by insertion from Π ($\Pi \Rightarrow_i \Pi'$).

In both cases we say that Π reduces Π' and write $\Pi \Rightarrow \Pi'$ if either $\Pi \Rightarrow_e \Pi'$ or $\Pi \Rightarrow_i \Pi'$.

The reduction relation and target proofs are related in the following sense (see [11] [26] for detailed proofs).

Theorem 3.1 *If* Π_1 *is a target proof for a given sequent* s *and* $\Pi_1 \Rightarrow \Pi_2$ *then* Π_2 *is also a target proof for* s.

The sequent $P \vdash g$ is provable if each branch of a target proof for $P \vdash g$ has a leaf of the form $P \vdash C, \top$.

An agent oriented approach to these construction leads us to recognize LO main mechanisms as *multiset transformation* and *broadcasting*. An LO program creates and manages a set of agents; they have their *own* life and their evolution is expressed by multiset rewriting, while communication is described by broadcasting. In more detail, each agent is a pair $< T, P >$ where T is a multiset of tokens and P is an LO-program.

Each agent tries to apply the program rules (Prolog like clauses) to the multiset it possess (its internal *state*). In this continuous game, with an expansion step, three recurrent patterns of rewriting can occur:

Transformation It is the case when the agent changes its multiset. This means that a program clause has been applied to a sub-multiset of the state multiset. Some tokens have been removed and others are added, so that the agent has changed its state, but it is also the case with the use of the @ rule, where we have an internal distribution of tokens.

Duplication In this case an agent is cloned. Of course both the multiset and the attached program are cloned.

Termination The agent disappears and dies. It means that the agent's task has been successfully finished.

Based on the so called *asynchronous* fragment of Linear Logic [31], an LO open-system evolution is represented by the search of a proof for a given goal, according to the programs attached to the different agents.

The natural correspondence between coordination concepts and proofs is as follows:

Agents	Proof Nodes
Transitions	Nodes Expansions
transformations	@ or $\circ\!\!-$
duplications	&
terminations	$\top$
Broadcasting	Insertion of a Formula
System Evolution	Proof Construction

Broadcasting a token t means, for a given agent α, to send t to all other agents β; it corresponds to the insertion of an atom t into each open node of the proof. Some of these aspects will be represented in a graphical manner in this work, when we base our analysis on the Interaction Abstract Machine metaphor ([9]). The broadcasting mechanism takes LO away from its logical basis, so that linear logic is no more a suitable semantics for LO.

For each goal formula G, the $||G||$ set of its @-components (read *par components*) is inductively defined as follows:

Definition 3.2 *Each @-component of G is a multiset of atoms, corresponding to a single branch in a proof of G.*

- $||A|| = \{\{A\}\}$ *for each atom* A
- $||G@H|| = \{G_i \cup H_j : G_i \in ||G|| H_j \in ||H||\}$
- $||G\&H|| = ||G|| \cup ||H||$
- $||\top|| = \emptyset$

Based on ideas presented in [40, 23] and [46], we associate, for instance, each agent of the language to a monomial of the ring and each global state (which is a set of agents) to a polynomial following the correspondence which is informally outlined below:

polynomial rings	**agents**
variables	token(atoms)
monomials	@-components
polynomials	leaves of proof (set of @-components)
polynomial division	progression rule
polynomial $\lambda - \rho$	method $\lambda \circ\!\!- \rho$

The systems obtained by the interactions of autonomous and heterogeneous agents have their interpretation in the metaphor of the Interaction Abstract Machine (IAM). Introduced in [9], the main characteristic of the IAM is that of enabling interactions among independent subsystems, in order to capture the global behaviour of the system by the analysis of the single autonomous components.

Similarly to the Chemical Abstract Machine metaphor [18], a IAM describes the components of a system in terms of multisets and moves the idea of its construction from a pure form based on the chemical reaction to a social-biological analisys; further, it analyzes organizational behaviours in terms of suborganizational systems. In the IAM metaphor, the state of a single agent is represented as a set of particles evolving within a given computational space. In the section 5 we introduce the IAM as the most direct informal counterpart of the LO computational model and provide definitions and examples often in terms of LO programs. This kind of multiset rewriting is the key for our correspondence towards polynomial reduction, and we exploit their concurrent capabilities. When an agent A, represented by a polynomial $||A||$, activates n methods on n disjoint sets of resources, it means that a n-step reduction has been computed on $||A||$.

In this framework we also address the issue of how far can the debate about multiset rewriting be extended, and we study which is the way how polynomial reduction [21, 38] gives a semantics for agents in the style of LO. The approach uses a powerful structure called LO^C, in order to develop computations in terms of the combination of *rules* and *agents*, and states criteria with which manage proof search strategy. The target areas for our approach, Computer Algebra and Rewriting Systems, are well explored research fields, and many insights can be exploited, at least at the syntactical level, for coordinating activities. Parallelism and concurrency have had many applications in these areas, such as [33, 29], and future applications can be investigated in this direction.

The first introductory presentation of the basic algebraic tools for manipulating computational algebraic structure is now given, then general algorithms for computing Gröbner

basis of polynomial ideal are presented. The input to these algorithms is an ideal specified by a finite set of polynomials; the algorithms produce another finite basis of the ideal which can be used to reduce polynomials so that every polynomial in the ideal reduce to 0 and every polynomial in the polynomial ring reduces to a unique normal form.

The approach adopted here is based on the Rewrite Theory [36, 21, 23]. More exhaustive definitions can be found in [23, 30, 42, 43]. A polynomial ideal can be considered as an equational theory; its Gröbner basis is a complete rewriting system, when polynomials are viewed as rewrite rules, which in turn can be used to generate canonical forms for residue classes defined by the ideal on a polynomial ring. In this approach polynomials are simplified (reduced) using a single polynomial at a time. New polynomials are added to complete a basis, computed between pairs of polynomials in the basis. These concepts are closely related to the critical pairs concept between rules in a term rewriting system ([40], [39]), there are however syntactical and semantical differences between polynomial reduction and term rewriting as pointed out by [23] and [15, 16]

4 Polynomial Rings

Let $E[x_1, x_2, ...x_n]$ be the polynomial ring over indeterminates x_i, where the coefficient of terms are taken from a Euclidean domain (or a field E). A term is a power product $\Pi_i x_i^k$ and the degree of a term is $\Sigma_i k_i$. An admissible ordering on terms is an ordering satisfying the following properties:

- $1 < t$ for any t term
- if $t < s$ then for any term p it results that $p \cdot t < p \cdot s$

Different total orderings on term can be defined; two of the most commonly used are the *lexicographic* and the *total degree* orderings, both well founded and admissible orderings.

We suppose that $<$ is one of the previously defined orderings on terms of $E[..., x_i, ...]$. It is possible to describe an ordering on the polynomials induced by $<$. In fact let $p = m + r$ be a polynomial such that the term of the monomial m is greater than those in r; then m is called the *head monomial* of p (written as $hm(p)$) and r is called the *rest* of the polynomial p. The head term of p is denoted by $ht(p)$ while the head coefficient will be denoted by $hc(p)$.

A well founded ordering on terms in $E[x]$ and a well founded ordering on E define a well founded ordering on polynomials of $E[x_1, ..., x_n]$ in a natural way:

- $0 < p$
- $p_1 < p_2$ if and only if either $ht(p_1) < ht(p_2)$ or $ht(p_1) = ht(p_2)$ and $hc(p_1) < hc(p_2)$, or $hm(p_1) = hm(p_2)$ and $p_1 - hm(p_1) < p_2 - hm(p_2)$

4.1 Polynomials as Rewrite Rules

Let $m_1 = c_1 t_1$ be the head monomial of a polynomial p. If $rest(p) = p - m_1$, then the rewrite rule corresponding to polynomial p is

$$c_1 t_1 \rightarrow -rest(p)$$

If p is a monomial then the right hand side of the rule is 0. In other words, the rule corresponding to p is constituted by a left hand side where a term t_1 is present and a right hand side constituted by a polynomial (eventually 0) $rest(p)$ whose head term is smaller (w.r.t. $<$) than t_1.

This rule is used to rewrite polynomials as follows: let m be a monomial, where $m = ct$ and $c \neq 0$. If the quotient obtained by dividing c by c_1 is 0 or there is any term σ such that $t = \sigma t_1$ then m cannot be rewritten by the rule $c_1 t_1 \rightarrow -rest(p)$. If $t = \sigma t_1$ then

$$m \rightarrow bt + a\sigma(-rest(p))$$

where $c = ac_1 + b$ and a and b are respectively the quotient and the remainder obtained by dividing c by c_1 using a division algorithm on E.

Let q be the polynomial $q_1 + ct$ having ct as head monomial that can be rewritten using the rule corresponding to p. Then

$$q \rightarrow q_1 + btt + a\sigma(-rest(p))$$

Example 4.1 *Let Z be the ring of the integers and $p = 2x^2y - yz$ be a polynomial in $Z[x, y, z]$. It corresponds to the rewriting rule $2x^2y \rightarrow yz$ and the polynomial $8x^3yz + xz^2 - y$ reduces to $4xyz^2 + xz^2 + y$ via p.*

When we consider a set of polynomials $P = \{p_1, p_2..., p_k\}$ then it is possible to extend the relation to P and say that a polynomial t reduces to a polynomial s (written as $t \rightarrow^P s$) if there is a finite sequence

$$t \rightarrow h_1 \rightarrow h_2.... \rightarrow s$$

of reduction via the elements of P.

Since a division algorithm always produces a remainder *smaller* than the element being divided when the quotient is non zero, using the well founded ordering defined on polynomials we have

Theorem 4.1 *Any finite subset P of a polynomial ring $E[x_1, ...x_n]$ induces a rewriting relation $\rightarrow^P$ which is Noetherian.*

The proof, making strong use of term orderings on polynomials, can be found in [54, 57]. We emphasize here that the reflexive, symmetric and transitive closure of the rewriting relation defined above, and denoted by $\rightarrow^*$, is equivalent to the ideal congruence relation, which is a standard relation when considering rings and ideals. We are going to give the definition of congruence in polynomial rings even if it is a definition used for rings in general.

Definition 4.1 *Let $B = \{b_1, b_2, ..., b_k\}$ be a finite set of polynomials. $I \subset E[x_1, ..., x_n]$ is an ideal of the polynomial ring $E[x_1, ..., x_n]$ induced by B if*

$$I = \{p : p = \Sigma_i q_i b_i \ / b_i \in B\}$$

for some $q_i \in E[x_1, ..., x_n]$

Each ideal I (and usually written as $I = (b_1, ..., b_k)$) induces a congruence relation ($\cong_I$) on the ring $E[x_1, ..., x_n]$, which can in turn *splitted* into lateral disjoint classes:

Definition 4.2 *Let p and q be two elements of $E[x_1, ..., x_n]$, and $I = (b_1, ..., b_k)$ an ideal with $b_1, ..., b_k$ as a base. We say that $p \cong_I q$ if and only if*

$$p = q + \Sigma_i q_i b_i$$

for some $q_i \in E[x_1, ..., x_n]$

In this case p and q are in the same lateral class, usually denoted by means of a representative element $[p]$.

Classical algebraic results state that the reflexive, symmetric and transitive closure of $\rightarrow_B$ can be identified with the ideal congruence $\cong_B$ (for any subset B).

Theorem 4.2 $\rightarrow_B^* = \cong_B$

A basis B for an ideal $I = (b_1, ..., b_k)$ can describe different reductions for polynomials depending by the kind of division we are operating and corresponding to a form of non determinism when computing with *general* basis of ideals.

Example 4.2 *Let $A[x, y, z]$ be a polynomial ring over the indeterminate x, y, z and $f_1 = x^3 - yz$, $f_2 = xz - y^2$, $f_3 = x^2y - z^2$ three polynomials constituting a basis F for an ideal $I = (F)$. The monomial x^2yz can be rewritten to xy^3 or to z^3 depending on the use of f_2 or f_3 as reduction rules.*

The previous example shows that the monomial x^2yz has no unique normal form with respect to the basis F. In this case we have two distinct normal forms (always in the same lateral class) depending on how the monomial is rewritten. This is the main motivation for the introduction of Gröbner basis.

A Gröbner basis G is in fact defined as a basis for an ideal I such that for any polynomial $q \in E[x_1, ..., x_n]$, no matter how q is rewritten using the rules corresponding to polynomials in G, the result is always the same (i.e. unique). It can easily be shown that this definition implies that for any polynomial p in the ideal I generated by G, it results $p \rightarrow_G^* 0$.

Let us analyze the previous example again.

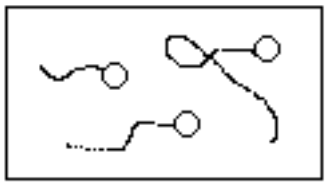
Example 4.3 *We consider the ring A and the ideal $I = (f_1, f_2, f_3)$. A Gröbner basis for I is $\{f_1, f_2, f_3, f_4, f_5\}$ where $f_4 = xy^3 - z^3$ and $f_5 = y^5 - z^4$. It is possible to test with some examples that the previous declared property holds.*

5 Interaction Abstract Machine

In either Gamma and LO, or following the Chemical Metaphor [18], the elements of the multiset on which to operate, can be thought of as molecules freely moving, imitating the chemical solution behaviour. The molecules react with each other according to some reaction rules. For instance in LO, a rule (method) keeps in its body a part constituting the reaction condition, and another one that is the action condition.

When a bag of molecules satisfy the reaction condition, at that time it can be rewritten in the way that the action condition states.

Each particle represents a source and it is assumed that they are in perpetual *Brownian* motion, and may randomly collide. Graphically, the evolution space (computational space), representing the agent state, is pictured by a box where particles are in perpetual motion. Particles are circles with names and their motion is represented by a line; collision points are finally drown with black circles that make old resources disappear and new ones appear.



5.1 Single Agent Systems

We first analyze the behaviour of a single agent system; even at this level, this gives us useful hints and several points of discussion about features and properties of a metaphor like the IAM.

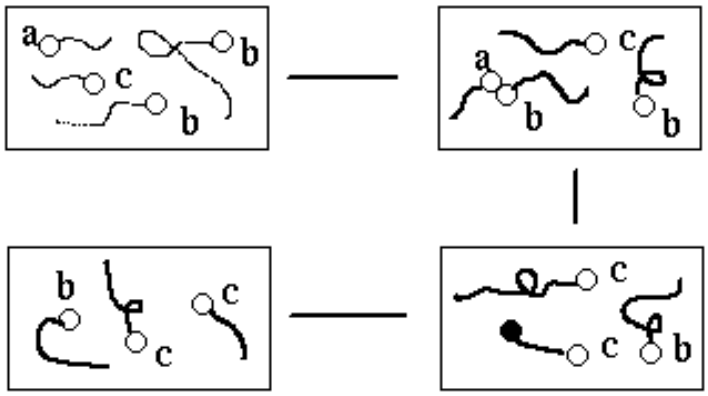
Methods in this metaphor are represented by two sided laws (it is possible to think now simply to LO methods) governing particle interactions during the collision. A single agent (the state of a single agent) is represented by a multiset of particles (resources, tokens) and when using a method some particles disappear, while others are created. For example $\{a, a, b, c\}$ could be a state attached to an agent.

$$a_1@a_2...@a_n \circ\!\!- b_1@b_2...b_m$$

is a method where b_i and b_j are resources. Here $a_1@a_2...@a_n$ is called the "head" of the method while $b_1@b_2...b_m$ is the "body".

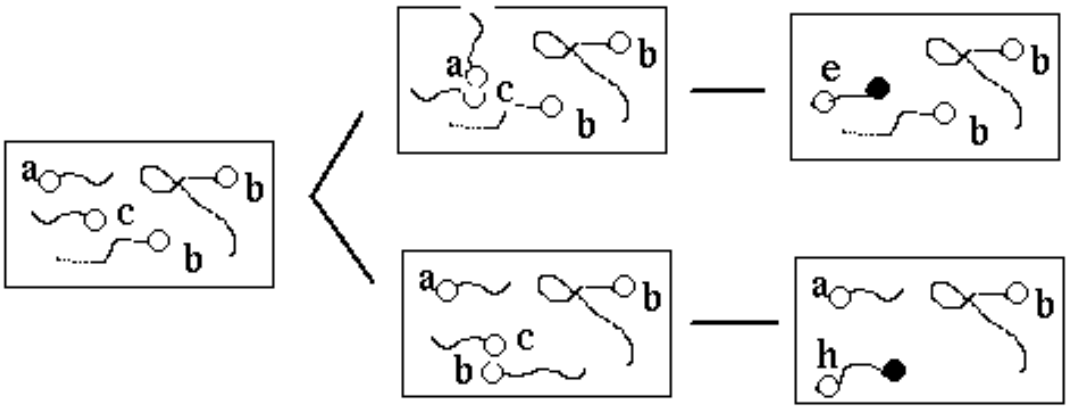
When the method is activated by the agent, then, in the agent state, there is a replacement of the bag of tokens matching the head of the method. A new state is in fact obtained replacing the old resources in the head with those in the body of the methods. The remaining part of the state is left unchanged.

Example 5.1 *Let $A = \{a, b, b, c\}$ be an agent and $a@b \circ\!\!- c$ a method. In this case the transition leads us to the new state $A' = \{b, c, c\}$ since the tokens a and b of A have been replaced by c. Graphically, the example can be pictured by*



With such a system it is possible to model a blackboard system: the agent (state of the agent) corresponds to the blackboard while methods represent the different sources of knowledge interacting through the blackboard. Of course the left hand side of a rule specifies tokens which are removed from the blackboard, conversely their right hand side identifies with tokens are added to the blackboard.

In this case (single agent) the behaviour of a IAM suitable describes the CHAM metaphor [18] where the evolution laws are interpreted in terms of chemical reactions. The following picture shows the competitive behaviour that a IAM can describe.



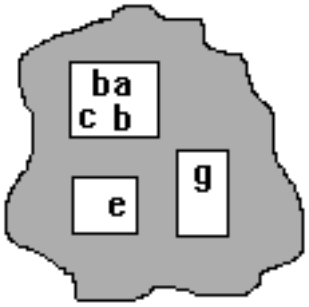
Typically a IAM agent has an internal competitive behaviour in the sense that if two methods conflict, then in the agent state it is possible to activate only one of the two. Conflicts between two agents mean that their left hand sides both match with a bag in the state. In this example two methods such as $a@b \leftarrow e$ and $a@c \leftarrow d$ are in conflict with respect to the agent state $\{a, b, c\}$. Now it results that the agent will progress in a completely nondeterministic manner. An exclusive choice is imposed and it is the random motion and the consequently random collision among particles that *manage* the method triggering. Since the application of a method consumes the resources, not both methods can be applied, the competitive behaviour among methods introduces a form of OR-concurrency, besides nondeterminism. In this respect a single agent can be thought of also as a tool for creating competing alternatives, like humans do when they face everyday life tasks.

5.2 Multiagents System

Let now describe a multiagents system: to this aim we need to define an environment in which it is possible to create, terminate and evolve agents. The primitives for these operations are functions enhancing the possibilities of the previous defined single agent transformation functions. Methods are in fact more powerful tools and their syntax is indeed extended in this way: the left hand side of a method remains defined as before but the right hand side (body) of a method may now be formed by 0 or more multisets of tokens, instead of a single multiset. Following [9] we denote by $\top$ an empty body and use the & symbol to connect two or more multisets of tokens. As in the single agent case, a method can be triggered by an agent a if the resources of the head of the method are present in its state. Three cases are now possible for an agent when triggering a method m:

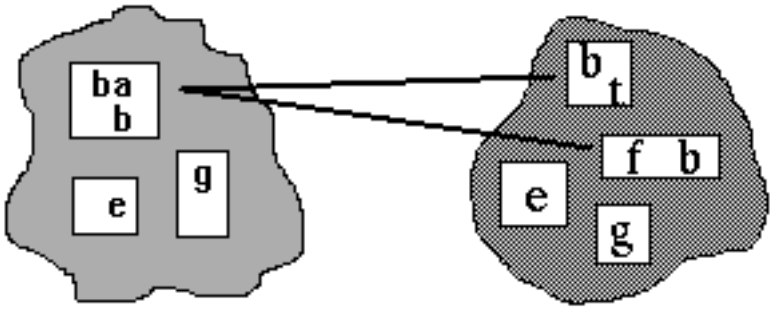
1. if the right hand side is the empty-set $\top$ then we have the termination of the agent .
2. if the body of the method contains only a multiset of resources like $g@h@...k$, i.e. without the & connectives, then, as in the single agent case, the body replaces the head in the agent state.
3. if the body has more, say n, multisets of resources, then $n-1$ copies of the agent are generated and in each of the n copies the head of the method is replaced by the corresponding multiset of the body.

We give a pictured example in which the boxes represent the agents with tokens, while the global environment is the gray area all around the agents.



Each agent, by the use of the third rule, becomes completely autonomous w.r.t. the others in the environment, and starts its own life in a sort of Unix *fork* command. An agent created in this manner can now locally transform itself without affecting the others; the system is completely dynamic, since new agents can appear while others disappear at any time. It is important to notice that such a multiagent system describes another form of parallelism different from the one previously analyzed in the single agent case, and called OR concurrency. While the first kind of parallelism arises as a form of competitive interaction among threads within an agent, this kind of parallelism enables a kind of AND concurrency, given by the creation of autonomous agents capturing the situation of independent, loosely integrated subsystems. In this respect, the IAM metaphor leads, to an improvement of classical agent oriented models of computations, such as Actors [2].

Note, in fact, that the OR concurrency embodied in the @ connectives among tokens, accounts for a sort of internal distribution of subtask. This means that we can *wrap* together in a single process several different subprocesses each of which may represent a possible option for the evolution of the overall process. On the other hand the AND concurrency embodied in the conjunction & accounts for a kind of external cooperation.



Here the main idea is that the metaphor enriches the model of computation, with a notion of "structured process" capable of supporting an *organizational* style of programming [12]. Furthermore OR concurrency is different with respect to other forms of disjunctive reasoning previously available in logic programming languages. It is different, for instance, from the disjunction available in some Prolog implementation, where a disjunctive goal succeeds if some subgoal separately succeeds. Finally we emphasize that those resources, of an agent state, which are not implied in a step of transformation, remain part of the new state and may be involved in another step of process propagation at a further stage of the agent evolution. This is fundamental in OR concurrency: elements in the state associated to an agent can lie dormant for a while but may at any time reenter the computation.

A conceptually different notion of multiple independent subsystems is also achieved with respect to the CHAM metaphor, where the concept of *membranes*, encloses sub solutions in a CHAM. Here the difference is that a IAM has a completely flat structure. This means that the agents are all at the same level and are not nested as in a CHAM, where solutions can be enclosed within other solutions up to an arbitrary depth. This feature has important consequence in terms of our mapping between coordination languages and polynomial rings, where a flat structure like the polynomial rings is chosen as codomain for our correspondence.

In such a model, inside an agent we have a form of interdependence induced by the production - consumption of resources: consumers (methods) depend on producers for satisfying their needs and a competition arises as a consequence of the same resource being usable by two or more consumers. Now agents are closed entities and their interactions must be managed by a form of explicit communication; to do this an elementary and powerful form of speech acts is provided. The traditional message passing scheme relying on addresses does not apply to the IAM since the notion of agent's address is lost. When an agent is created, it does not have a name, an address or any sort of identity. It is a *collection* (@) of resources directly inherited from the agent it was cloned from. The communication is, in this case, managed by a form of broadcasting. Agents who are listening on some channel receive information broadcast on such channels by other agents. If the message satisfies certain conditions, then the agent reacts consequently, otherwise it simply ignores the message. This kind of communication, based on broadcasting, is

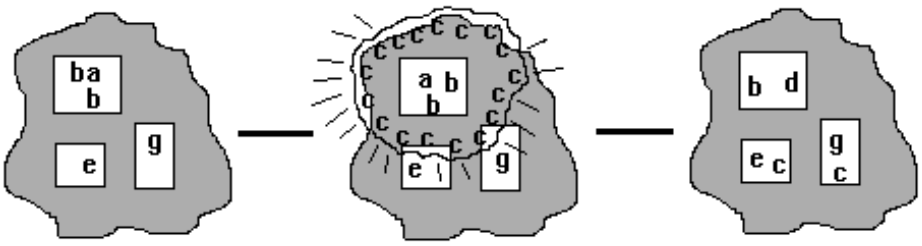
called *property driven* communication. It appears as a nice feature in providing support for open systems since it enables a strong form of dynamic reconfigurability. The desire of making agents communicate directly on the basis of their properties, rather than their name, permits to send requests away without specifying an address, but simply requiring whoever is in *charge* to process them.

More precisely, broadcasting is obtained by the extension of the syntax of methods: in the head of a method some resources can be *marked* with a special 1-ary symbol ˆ called *broadcast* marker. It is operationally used in this way: if the all non-marked resources of the head of a method m are found in an agent state a, then

1. a copy of the marked resources are added to the states of all the agents in the global system (a included)

2. the agent a that has activated the method m, evolves by the application of the method (without considering the marker ˆ)

The two steps above happen simultaneously: when a message is broadcasted there is no need to lock the whole system, add the marked copy to all agents and then unlock the open system. It can be shown that broadcasting is captured by the IAM metaphor introducing *waves* spreading across the agents. Each wave represents a resource (marked resource) that hits the agents states propagating to the whole system. Graphically, it is represented by a closed curve (the wave front) labeled with the name of the marked token.



Finally it is interesting to notice the difference between this mechanism of interaction induced by the broadcasting communication and the competitive interaction among different threads within an agent: during broadcasting, several distinct copies of a token are distributed to different agents, who are free to make different use of the copy. Broadcasting induces different responses to shared stimuli and a form of *cooperation* can be exploited. It can model the situation where complex problems need to be viewed by different cooperating experts, each providing a specific response to a common problem.

The IAM is indeed characterized essentially by two kind of behaviours: *internal distribution*, the situation when a complex task (single agent) is dealt with by partitioning it in several subtasks (multi agents) distributed to different parts, and *external cooperation*, the situation when an agent ask the cooperation of another agent. In the next section, we make use of polynomials as global state for a coordination system, while monomials correspond to agents.

6 LO^C: LO on par component

We first enrich the set of par components, previously defined, and introduced in [9], with a ring structure inherited from the Linear Object connectives. In order to do this we first

need to define the set of *co-agents* as a sort of complementary sets of agents.

- $M(A)$, the *positive goal formulas set*, is the set of all the multisets in A

$$M(A) = \{\{a, a, c\}, \{a, b, b\}, \{a\}, \emptyset, \{a, a\}, ...\}$$

  Its elements are called *agents*. They constitute the @-components of the LO language ($||g||$ for each goal formula g).

- for $x \neq \emptyset \in M(A)$ $-x$ denotes its *co-agent* (constraint agent).

- $-M(A)$ denoting the set of co-agents.

Now we can define the carrier set LO^C.

Definition 6.1 *The set $LO^C = P(M(A) \cup -M(A))$ is the power-set of the union of $M(A)$ and $-M(A)$. The following binary operations are defined:*

1. *if c_1 and c_2 are in LO^C then $c_1 \& c_2$ is in LO^C and it is given by a refinement on multisets union, i.e. $c_1 \& c_2 = \{x : x \in c_i \text{ and } \forall y \in c_j \ (y \neq -x) \text{ with } i \neq j\}$*

2. *if c_1 and c_2 are in LO^C then $c_1 @ c_2 = \{x \cup y : x \in c_1, y \in c_2\}$*

The domain LO^C with the two defined operations is a commutative ring with unit, since it enjoys the following properties:

- the empty-set is the identity for &. It will be also denoted by 0.

- $1 = \{\{\}\}$ is the unit element for @

- $-a$ is the co-agent of a and $a \& (-a) = \emptyset$ for all a

- & is associative and commutative

- @ is associative and commutative

- @ distributes with respect to & i.e. $a @ (b \& c) = a @ b \& a @ c$

- $-(-a) = a$ for each $a \in LO^C$

- LO^C is an Euclidean domain in fact

$$a \neq 0 \ b \neq 0 \ \Rightarrow \ a @ b \neq 0$$

Here we prove that the above structure is a ring.

Lemma 6.1 *$(LO^C, \&, -, \emptyset)$ is a commutative group*

Proof 6.1 *1. The empty-set is the unit for & since for a given $x \in LO^C$ it is*

$$\begin{aligned} x \& \emptyset &= \{x_i : \ x_i \in x \ \wedge \ \forall y \in \emptyset \ (x_i \neq y)\} = \\ &= \{x_i : \ x_i \in x\} = x \end{aligned}$$

2. $\&$ *is associative since for each* x, y *and* z *in* LO^C *we have*

$$\begin{aligned}(x\&y)\&z &= (\{x_i \in x \ : \ \forall y_j \in y(x_i \neq -y_j)\} \cup \\ &\quad \{y_j \in y \ : \ \forall x_i \in x(-x_i \neq y_j)\}) \cup \\ &\quad \{z_k \in z \ : \ \forall w_h \in x\&y(z_k \neq -w_h)\}\end{aligned}$$

Each w_h *corresponds to a particular* x_i *or* y_j *and we can replace the three sets with*

$$\begin{aligned}&\{x_i \in x \ : \ \forall w_k \in y\&z \ (x_i \neq -w_k)\} \cup \\ &(\{y_j \in y \ : \ \forall z_k \in z(z_k \neq -y_i)\} \cup \\ &\{z_k \in z \ : \ \forall y_j \in y(z_k \neq -y_j)\}) = \\ = \ & x\&(y\&z)\end{aligned}$$

3. $\&$ *is trivially commutative as it is a restriction of classical multiset union*

4. *For each* $x \in LO^C$ $-x = \{-x_i \ : x_i \in x\}$ *is the inverse of* x *with respect to* x. *In fact*

$$\begin{aligned}x\&-x &= \{x_i : \ x_i \in x \ \wedge \ \forall x_j \in -x \ (x_i \neq x_j)\} = \\ &= \{\} = \emptyset\end{aligned}$$

Lemma 6.2 *The structure* $(LO^C, @, 1)$ *is a commutative monoid.*

Proof 6.2 *It is trivially proved that* $@$ *satisfies the associative and commutative properties since it corresponds to multisets union. Now if* $x \in LO^C$ *it results that*

$$\begin{aligned}x@1 &= \{x_i \wedge \emptyset \ : x_i \in x\} = \\ &= \{x_i \ : x_i \in x\} = \\ &= x\end{aligned}$$

Theorem 6.1 $(LO^C, \&, @, -, 0, 1)$ *is a commutative ring with unit.*

Proof 6.3 *We need to prove only that the distributive law holds*

$$a@(b\&c) = a@b\&a@c$$

Let us denote with a_k, b_i *and* c_j *respectively the elements of* a, b *and* c: *we have that*

$$\begin{aligned}a@(b\&c) &= a@(\{b_i \in b, c_j \in c/ \ b_i \neq -c_j\}) = \\ &= \{b_i \cup a_k, \ c_j \cup a_k/ \ b_i \neq -c_j \wedge a_k \in a\} = \\ &= \{b_i \cup a_k, \ c_j \cup a_k/ \ (b_i \cup a_k) \neq -(c_j \cup a_k)\} = \\ &= \{b_i \cup a_k/ \ (b_i \cup a_k) \neq -(c_j \cup a_k)\} \cup \{c_j \cup a_k/ \ (c_j \cup a_k) \neq -(b_i \cup a_k)\} = \\ &= \{x \in a@b/ \ \forall y \in a@c \ x \neq -y\} \cup \{y \in a@c/ \ \forall x \in a@b \ x \neq -y\} = \\ &= (a@b)\&(a@c)\end{aligned}$$

and the theorem is proved.

Example 6.1 *Let* $B = \{\{a\}, \{a, b, c\}\}$, $C = \{\{a, a, c\}, -\{a, b\}\}$ *and* $D = \{\{a, c\}, \emptyset\}$ *three* LO^C *elements. Then*

$$C \& D = \{\{a, a, c\}, -\{a, b\}, \{a, c\}, \emptyset\}$$

$$C @ D = \{\{a, a, a, c, c\}, -\{a, a, b, c\}, -\{a, b\}, \{a, a, c\}\}$$

With respect to the distributive law,

$$\begin{aligned} B @ (C \& D) &= \{\{a\}, \{a, b, c\}\} @ (\{\{a, a, c\}, -\{a, b\}\} \& \{\{a, c\}, \emptyset\} \\ &= \{\{a\}, \{a, b, c\}\} @ \{\{a, a, c\}, -\{a, b\}, \{a, c\}, \emptyset\} \\ &= \{\{a, a, a, c\}, -\{a, a, b\}, \{a, a, c\}, \{a\}, \\ &\quad \{a, a, a, b, c, c\}, -\{a, a, b, b, c\}, \{a, a, b, c, c\}, \{a, c, b\}\} = \\ &= \{\{a, a, a, c\}, -\{a, a, b\}, \{a, a, a, b, c, c\}, -\{a, a, b, b, c\}\} \\ &\quad \& \{\{a, a, c\}, \{a\}, \{a, a, b, c, c\}, \{a, c, b\}\} = \\ &= (B @ C) \& (B @ D) \end{aligned}$$

We remark that $\emptyset$ *annuls w.r.t.* @, *we mean that for all* $C \in LO^C$ *it results* $C @ \emptyset = \emptyset$

The elements of LO^C will also be called global states terms.

Finally we observe that the restriction of & to $P(M(A))$, the *positive goal formula set* of LO^C, is equivalent to $\cup$ and returns $(P(M(A)), \&, \emptyset)$ that is a commutative monoid. The previous statements describe the shared evolving sets of agents as a module, more precisely it is a ring given by the action of a commutative monoid $(LO^C, @, 1)$ over a commutative group $(LO^C, \&, 0)$ with $-$ as inverse.

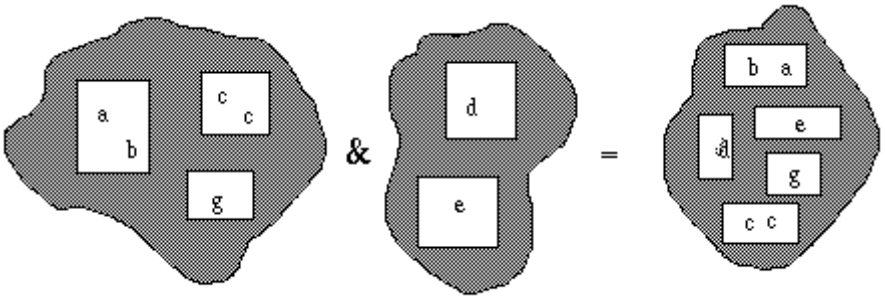
We give now more examples where agents (or better set of agents) are interpreted as elements of the LO^C ring emphazising the polynomial interpretation for these sets of agents. If we, for instance, define 2 as $\{\{\}, \{\}\}$ and a^2 as $\{\{a, a\}\}$, a polynomial interpretation is a direct consequence: the examples given below clarify this point.

rings	**agents**
$a + c$	$\{\{a, c\}\}$
1	$\{\{\}\}$
$(a \cdot a) + b$	$\{\{a, a\}, \{b\}\}$
$a^3bc + ac + b$	$\{\{a, a, b, c\}, \{a, c\}, \{b\}\}$
$c \cdot (a^2 + b)$	$\{\{a, a, c\}, \{c, b\}\}$

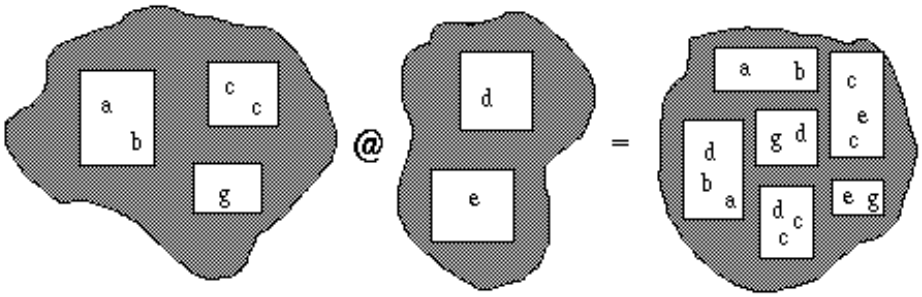
The last correspondence derives observing that

$$c \cdot (a^2 + b) = \{\{c\}\} @ \{\{a, a\}, \{b\}\} = \{\{a, a, c\}, \{c, b\}\}$$

The & operation is graphically described by the following picture:



while @ is represented by the following:



Here we briefly rephrase some well known algebraic definitions that are used in the next sections. The definitions are given in terms of the ring LO^C.

Definition 6.2 *The ideal(P) is the subset of LO^C of $\&$−combinations of @-components of P elements:*

$$ideal(P) = \{f \in LO^C : f = \&_i(p_i @ h_i)\}$$

where p_i's are in P, and the h_i's are elements in LO^C.

An ideal induces a binary relation (a congruence) in LO^C.

Definition 6.3 *If $A, B \in LO^C$ and $A = B \&_i(P_i @ Q_j)$ with $P = \{P_1, ...P_k\}$ a program and Q_j elements of LO^C, then we say that A and B are congruent modulo P, denoted by $A \leftrightarrow B$.*

7 Semantical interpretation

In this section we describe the $||\cdot||$ correspondence from objects of a IAM into LO^C, then we will make use of this *mapping* when proving the correctness of the calculus with respect to LO.

7.1 Interpretation of Goals

We define $||\cdot||$ as the mapping that associates elements of the language LO to @-components of LO^C. The interpretation is given in terms of the par component set. It is inductively defined as

$$\begin{aligned} ||a|| &= \{\{a\}\} \\ ||g@h|| &= ||g|| @ ||h|| \\ ||g\&h|| &= ||g|| \& ||h|| \end{aligned}$$

for any atom a and LO goal formulas g and h.

As already seen, LO^C is the domain of interpretation of *par*-components. This is a direct consequence of the induced operations, satisfying $||G|| \& ||H|| = ||G \& H||$ (for G and H positive goal formulae).

7.2 Interpretation of Methods

Here we extend the @-component construction, for global states, to LO methods and, in order to do this, we need to introduce the co-agents, representing a sort of constraints on a global state.

As it has been stated, $(LO^C, @, \&)$ is a ring. Informally it represents the levels of a proof in LO; each element is in fact a set of leaves in a LO-proof (or better a multiset of leaves) and composing two sets of leaves represents communications between agents. We now describe more precisely which kind of communication can be well represented in such a ring: we assume that beyond agents (multiset of agents)in LO^C, there are co-agents (multisets of co-agents).

Example 7.1 *Observe first that if a global state $s \in LO^C$ contains co-agents, then it has this representation:*

$$\{\{a, a, c\}, \{a, b, b\}, -\{a, b\}\}$$

now for each state s' it is not possible that $s \Rightarrow s'$ and $\{a, b\} \in s'$. Here the $\Rightarrow$ denotes a one step evolution of proofs but we use it like if each proof is represented by a global state in LO^C.

Of course for some s'' it is allowed that $s \Rightarrow^* s''$ and $\{a, b\} \in s''$. So the meaning of a co-agent is that it forces the "next" possible status to be a state without the agent (associated to the co-agent). Algebraically it is, the inverse of an element a with respect to &. For example given

$$\{\{a, a, c\}, \{a, b, b\}, -\{a, b\}\}$$

and

$$-\{\{a, a, c\}, \{a, b, b\}, -\{a, b\}\}$$

their & composition is $\emptyset$.

The interest about these objects is in that they are well suited to represent methods in our semantics. A method $A \multimap B_1 \& B_2 \& ... B_n$ in which both A and B_i are atoms @-connected, can be represented by an element of LO^C considering $\multimap$ as the unary inverse operation $-$, with respect to & of LO^C. The aim is to think of $M(A)$ only as a starting point, where we should add methods interpretations $||m||$ and finally obtain LO^C.

Definition 7.1 *An LO method $M = A \multimap B_1 \& B_2 \& ... B_n$ has the following representation*

$$||M|| = \{\{A\}, -\{B_1\}, -\{B_2\}, ...\}$$

where each B_i is a co-agent.

Example 7.2 *Let $a@b@c \multimap d\&(e@f)$ be a method. Its representation following $||\cdot||$, is $\{\{a, b, c\}, -\{d\}, -\{e, f\}\}$*

It is easy now to describe what kind of object describes a program P of the IAM metaphor in the LO^C framework.

Definition 7.2 *Let $P = \{p_1, p_2, ..., p_n\}$ be a set of methods, than*

$$||P|| = \{||p_1||, ||p_2||, ..., ||p_n||\}$$

where for each i, $||p_i|| = \{\{A^i\}, -\{B_1^i\}, -\{B_2^i\}, ...\}$.

7.3 Interpretation of Proofs

We recall that for a program P and a goal formula g, it is possible to find a set of target proofs $\Pi_{P,g}$ for the sequent $P \vdash g$ in LO. We extend the mapping $||\cdot||$, with codomain LO^C to target proofs, considered as pairs of elements in the IAM metaphor.

Definition 7.3 *If Π is a target proof of $\Pi_{P,g}$, $||\Pi||$ is the union of the @-components of its leaves.*

Example 7.3 *Let Π be the following proof:*

$$\dfrac{P \vdash a \qquad \dfrac{P \vdash h@h}{P \vdash g}}{P \vdash a\&g}$$

Its interpretation in LO^C is $\{\{a\},\{h,h\}\}$

It results that the transformation of a proof by means of a structural decomposition inference figure, does not change the interpretation in LO^C.

Lemma 7.1 *For each target proof Π, if $\Pi \Rightarrow \Pi'$ via the structural rules, then $||\Pi|| = ||\Pi'||$*

Proof 7.1 *Let $\Pi \Rightarrow \Pi'$ and $\{A_1, A_2, ..., A_n\}$ be the set of the leaves of Π. We may assume, without less of generality, that Π' is obtained from Π by use of a structural rule on the agent A_i, then this means that the agent is not a flat goal. Rather, it is a formula with some @ or & connectives in its body; we obtain that the $||A_i||$ elements in LO^C directly correspond to the interpretation of the proof expansion of A_i. In fact:*

- *if $A_i = G@H$ then $||G@H|| = \{\{G,H\}\}$. Now the expansion Π_{A_i} of A_i has one new leaf labeled by G, H and its interpretation is given by the multiset union of the interpretation of its leaves. This means that $||\Pi_{A_i}|| = ||G,H||$ and this is equal to $\{\{G\} \cup \{H\}\} = \{\{G,H\}\}$*
- *if $A_i = G\&H$ then $||G\&H|| = \{\{G\},\{H\}\}$. Now the expansion Π_{A_i} of A_i has two new leaves labeled by G and H and its interpretation is given by the multiset union of its leaves interpretation. This means that $||\Pi_{A_i}|| = ||G|| \cup ||H||$ and this is equal to $\{\{G\}\} \cup \{\{H\}\} = \{\{G\},\{H\}\}$*
- *Each goal A_i of the form $\{a, b, ..\top\}$ terminates and this corresponds in our semantics to the fact that each element $x \in LO^C$ is equal to 0 when @-composed with $\top$*

Example 7.4 *Let $g = a\&b$ be a given goal formula and P a program then $||P \vdash a\&b||$ directly corresponds to $||g||$ and the interpretation of*

$$\dfrac{P \vdash a \quad P \vdash b}{P \vdash a\&b}$$

similarly has $||g||$ as an image.

Naturally, the interpretation changes when considering non structural rules, i.e. when a method m is triggered by an agent A, as in the example 7.3. Some nice properties hold anyway, and this will be the topic of the next sections. Sufficient conditions for the reduction relation $\Rightarrow$ between proofs will be expressed in terms of the operational semantics given below.

8 Operational semantics

We have given a semantics to the objects of LO, now a semantics of the calculus is proposed.

We first analyze how the relation $\Rightarrow$ between proofs is interpreted. The binary relation $P \vdash p \Rightarrow P \vdash q$ between LO proofs is induced by the congruence $||p|| \leftrightarrow ||q||$ modulo $||P||$ in LO^C

Theorem 8.1 *For any set of methods P and goals g and h it results that*

$$if\ P \vdash g \Rightarrow P \vdash h\ then\ ||P \vdash g|| \leftrightarrow ||P \vdash h||$$

Proof 8.1 *As a consequence of the Lemma7.1, if $P \vdash g \Rightarrow P \vdash h$, because of the use of structural rules only, then $||P \vdash g|| = ||P \vdash h||$ and trivially we obtain a congruence between the two LO^C objects $||P \vdash g||$ and $||P \vdash h||$. Let us assume that $P \vdash g \Rightarrow P \vdash h$ by the use of the method $g \circ\!\!- h$. Now $||P \vdash g|| = ||g|| = \{\{g\}\}$ and similarly $||P \vdash h|| = ||h|| = \{\{h\}\}$, which are congruent modulo $||g \circ\!\!- h||$ (and henceforth modulo $||P||$), since*

$$\begin{aligned} ||g|| &= ||h|| \mathbin{\&} (||g \circ\!\!- h||) = \\ &= \{\{h\}\} \mathbin{\&} \{\{g\}, -\{h\}\} \\ &= \{\{h\}, \{g\}, -\{h\}\} \\ &= \{\{g\}\} \end{aligned}$$

This directly corresponds in LO^C to the congruence modulo $||P||$.

Given P and g, respectively a program and a goal, they define and ideal in LO^C ($ideal(||P||)$) and a $||g||$ residing in a lateral class $[||g||]$ of the ideal generated by $||P||$. The use of decomposition inference figures enables us to write each target proof Π of $P \vdash g$ as $||g||$. If we make use of a method $\lambda \circ\!\!- \rho$ and apply a progression inference rule to a proof Π, then the image of Π', with $\Pi \Rightarrow \Pi'$, will be an element $||\Pi'||$ in the lateral class $[||g||]$ different from $||g||$, as we have seen that congruence modulo $||P||$ is a necessary condition to proofs reduction relation ($\Rightarrow$).

Lemma 8.1 *If g is a provable goal for the program $P = \{p_1, ...p_m\}$, then the global state $||g|| \in LO^C$ is congruent to 0 modulo $||P||$.*

Proof 8.2 *It is an easy consequence of the Lemma7.1.*

As a main result we have that

Theorem 8.2

$$P \vdash g \ \Leftrightarrow\ ||g|| \in ideal(||P||)$$

Proof 8.3 *We inductively prove that the decomposition inference figures satisfy the theorem:*

- *if $P \vdash G, H$ (if $||G||$ and $||H||$ are in the ideal generated by $||P||$) then $||G@H||$ (the $@$ composition is in the ideal$(||P||)$*

- *if* $P \vdash G$ *and* $P \vdash H$ *(if* $||G||$ *ed* $||H||$ *are in the ideal generated by* $||P||$*)then* $||G\&H||$ *(their* $\&$ *composition is in the ideal*$(||P||)$

- $P \vdash \{\top\} \cup C$ *means that each global state is equal to* 0*, when we have a* $@$ *composition with* 0*, and it belongs to ideal*$(||P||)$*.*

- *progression inference figures are instances of*

$$\frac{P \vdash C, G}{P \vdash C, A_1, A_2, ..., A_n}$$

where $A_1 @ A_2 @ ... @ A_n \multimap G$ *is a ground instance of a method in the program* P*. If we think of* LO^C *as a polynomial ring, it is possible to translate these inference into our framework as rewriting rules; each method* $A_1 @ A_2 @ ... @ A_n \multimap G$ *defines a polynomial* $a_1 @ a_2 @ ... a_n \& - (g)$ *and each polynomial defines a rewriting rule* $(a_1 @ a_2 @ ... a_n, g)$ *where* $a_1 @ a_2 @ ... a_n$ *and* g *are respectively the head side of the polynomial and the tail (the polynomial rest). In this way we can internalize the evolution concept,as it is established that evolution rules are polynomial (sets of agents) and it is possible to proceed with classical polynomial reduction.*

An intuitive description can be given by observing that each evolution induced by an LO method transforms positive global states into global states and does not allow the generation of co-agents. A method (λ, ρ) has its ρ component constituted by $\&$ collection of $@$-components and a replacement in a generic state s does not generate any negative agent[1].

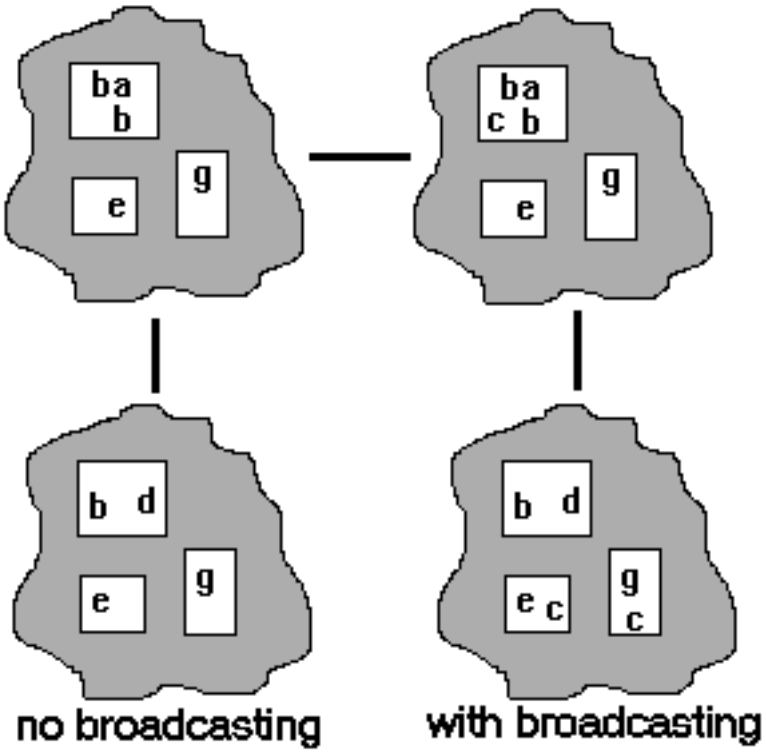
Example 8.1 *Let us consider the polynomial* $ab + a^2 + c$*. It represents the* LO^C *object* $\{\{a, b\}, \{a, a\}, \{c\}\}$*. A method* $a@b \multimap d\&e$ *enables the cancellation of the agent represented by* $\{a, b\}$ *and refresh the global state with a collection of new positive agents, that, in this particular case, are* $\{d\}$ *and* $\{e\}$*.*

Note that broadcasting a token means a unusual kind of polynomial division: the problem reside in the dynamical behaviour of the wave front of the broadcasted tokens, in fact, in the graphical example, with the rule $a@b@\hat{c} \multimap d$ the polynomial $ab^2 + e + g$ (representing the initial state), is first multiplied with the monomial of the broadcasted, then it is reduced with classical division algorithm.

$$bd + ec + gc = c(ab^2 + e + g) - (abc - d)b$$

In this respect the defined congruence does not hold, if we give a rough translation via the $||\cdot||$ mapping. On the other hand, only LO^C operations are necessary to represent methods with broadcasting, this enable us to manage it in LO^C and to investigate for extensions of classical polynomial congruence. A final remark on broadcasting is given graphically in the picture.

[1]Possible extensions to Linear Logic, and relations with other semantics, are under investigation. Actually the problem resides in a coherent interpretation of goals and methods with respect to the Linear equation $w \multimap a\&b\&c...\&f = w@(a\&b\&...\&f)^{\perp} = w@(a^{\perp} \oplus b^{\perp} \oplus ...f)$



9 Conclusions

In this paper an interpretation that characterizes the declarative flavor of coordination models, such as LO, Gamma and Linda, is proposed, considering their associative access to data. Although coordination is not limited to generative communication via shared data spaces, in this thesis we focused on shared data-space models since we are interested in their reactive behaviour. Within this approach, it is possible to describe "what" a programmer desires to compute while forgetting "how" to compute.

In this work an approach has been followed to model the development of calculi in these framework. It covers a pure algebraic semantics and defines a ring structure suited to model coordination models based on the shared memory paradigm of communication. This kind of semantics captures the the internal logics of the systems in a static view. We have defined an axiomatic semantics reflecting the behaviour of agents involved in a coordination environment, thus opening the way to the possibility of deriving properties and relationships of coordinated objects in terms of pure algebraic facts. The approach shows the power of a structure called LO^C in order to develop computations in terms of the combination of *rules* and *agents*. We have stated criteria with which the proof search strategy can be managed. The semantical interpretation associating global states to polynomials allows us to use algebraic algorithms on ring ideals [21]. For a given program P it is possible to describe a coherent way to distribute it on the nodes on a net, by use of such algorithms, such that for a given goal g, the evolution of its target proofs ($P \vdash g$) is the same as ($P' \vdash g$) where P' is the distributed program on the net, as Theorem 8.1 states. Furthermore it is possible to eliminate, under some algebraic conditions, forms of unwanted nondeterminism.

Directions for future research include the following issues. A natural refinemenent of

the algebraic semantics can be foreseen: the statical structure of polynomial rings does not completely capture communications. We have seen at the end of Chapter 3, for instance, how the LO operation of broadcasting does not fit completely into polynomial congruence. Note also that not every object of LO^C represents an element of the LO model. For instance the polynomial

$$a^2b + ac - c$$

having two positive monomials is not considered an LO rule (method). In fact it should correspond to the method

$$a@a@b\&a@c \circ\!\!- c$$

A possible interpretation of methods of this form is a subject for future investigation, that can mainly be based on a recent paper [27], in which the COOLL model of coordination is presented, which includes new coordination and modularity mechanisms and has been designed as an extension of LO.

In order to furnish the adequate dynamic features to a polynomial ring, a traditional view is to consider a sheaf of rings as a ring which *continuously varies*. In particular, we intend to investigate the classical sheaf representation of a ring ([20]), a process which would allow us to replace the study of an arbitrary commutative ring with unit, such as for instance the ring LO^C defined in this thesis, by that of a family of local rings [3].

References

[1] V.M. Abrusci. *Seminari di Logica Lineare.* Laterza, 1992.

[2] G. A. Agha. *Actors: a model of concurrent computation in distributed systems.* MIT press, 1986.

[3] S. Ambler and D. Verity. Generalized logic and the representation of rings. *Applied Categorical Structures*, 1996.

[4] P. Ancillotti and M. Boari. *Principi e tecniche di programmazione concorrente.* UTET libreria, 1996.

[5] J.M. Andreoli. Logic programming with Focusing proofs in Linear Logic. *Journal oj logic and computation*, 2, 1992.

[6] J.M. Andreoli. Coordination il LO. In *Coordination Programming: Mechanisms, Models and Semantics*, 1996.

[7] J.M. Andreoli, U. Borghoff, and R. Pareschi. Constraint based knowledge brokers. In *Proc. 1st Int. Symp. on Parallel Symbolic Computation - PASCO 94*, 1994.

[8] J.M. Andreoli, U. Borghoff, R. Pareschi, and J. Schilichter. Constraint agents for the information age. In *J. Universal Computer Science 1-12*, 1995.

[9] J.M. Andreoli, P. Ciancarini, and R. Pareschi. Interaction Abstract Machines. In *Research Direction in Concurrent Object Oriented Programming*, 1993.

[10] J.M. Andreoli, S. Freeman, and R. Pareschi. The coordination language facility: coordination of distributed objects. Technical report, Rank Xerox Research Center, Grenoble lab. Fr, 1995.

[11] J.M. Andreoli, L. Leth, R. Pareschi, and B. Thompsen. True concurrency semantics for a linear logic programming language with broadcast communication. In *Theory and pratice of software developments, TAPSOFT 93*, 1993.

[12] J.M. Andreoli and R. Pareschi. LO and behold! concurrent structured processes. In *OOPSLA-ECOOP 90, Ottawa, Canada*, 1990.

[13] J.M. Andreoli and R. Pareschi. Communication as fair distribution of knowledge. In *OOPSLA-91, Phoenix, Az, U.S.A*, 1991.

[14] F. Arbab. The IWIM model for coordination of concurrent activities. *LNCS*, 1061, 1996.

[15] G. Balestreri. External and internal rewriting. Technical report, CNR-IAC. Rome, 1992.

[16] G. Balestreri and G.F. Mascari. Concurrent rewriting and knuth-bendix completion theorem in a 2-category. Technical report, CNR-IAC. Rome, 1992.

[17] J.P. Banatre and D. LeMetayer. Programming by multiset transformation. *Communication of the ACM*, 36, 1993.

[18] G. Berry and G. Boudol. The Chemical Abstract Machine. *Theoretical Computer Science*, 96, 1992.

[19] D.G. Bobrow. Dimensions of interaction. *AI Magazine*, 1991.

[20] F. Borceaux. *Handbook of Categorical Algebra 3. Categories of Sheaves*. Cambridge University press, 1989.

[21] B. Buchberger. A criterion for detecting unnecessary reductions in the construction of gröbner bases. In *EUROSAM*, volume 72 of *Lectures Notes in Computer Science*, 1979.

[22] B. Buchberger. A critical pair completion in reduction rings. Technical report, Univ. of Lintz, Math. Inst. TR: CAMP-83-21.0, 1983.

[23] B. Buchberger and R. Loos. Algebraic simplification. In *Computer Algebra. Symbolic and Algebraic Computations*. Springer Verlag, 1983.

[24] N. Carriero and D. Gelernter. Linda in context. *Communication of the ACM*, 32, 1989.

[25] N. Carriero and D. Gelernter. Coordination languages and their significance. *Communication of the ACM*, 35, 1992.

[26] S. Castellani and P. Ciancarini. Comparative semantics of LO. Technical report, Dep. Comp. Science University of Bologna Italy, TR-UBLCS-94-7, 1994.

[27] S. Castellani and P. Ciancarini. Enhancing coordination and modularity mechanisms for a language with objects as multisets. In *Coordination Languages and Models*, volume 1061, 1996.

[28] P. Degano, R. DeNicola, and U. Montanari. A distributed operational semantics for CCS based on condition/event systems. *Acta Informatica*, 26, 1988.

[29] N. Dershowitz and J.P. Jouannaud. *Rewrite Systems*, chapter vol B. Handbook of Theoretical Computer Science. North Holland, 1990.

[30] F. Gadducci. *On the Algebraic Approach to Concurrent Term Rewriting.* PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, 1996.

[31] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50, 1986.

[32] J.Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge University Press, 1989.

[33] J.A. Goguen, C. Kirchner, and J. Meseguer. Concurrent term rewriting as a model of computation. *LNCS*, 279, 1987.

[34] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[35] A.A. Holzbacher. Coordination of distributed and parallel programs in concoord. In *Coordination Programming: Mechanisms, Models and Semantics*, 1996.

[36] G. Huet and D. Oppen. Equations and rewrite rules: A survey. *Formal Languages: Perspectives and Problems*, 1980.

[37] http://www.javasoft.com.

[38] A. KanriRodi and D. Kapur. Computing a gröbner basis of a polynomial ideal over a euclidean domain. *Journal of symbolic computation*, 6, 1988.

[39] J.W. Klop. Term rewriting systems. Technical report, CWI Amsterdam, 1990.

[40] D.E. Knuth and P. Bendix. Simple word problem in universal algebra. In J. Leech, editor, *Computational Problems in Abstract Algebra.* Pergamon Press, 1967.

[41] Y. Lafont. Introduction to linear logic. Lectures Notes for Sumer School on Constructive Logics and Category Theory. Isle of Thorns, 1988.

[42] J. Meseguer. Rewriting as a unified model of concurrency. Technical report, SRI CSL-93-02R, 1990.

[43] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical computer science*, 96, 1992.

[44] J. Meseguer and U. Montanari. Petri Nets are Monoids. *Information and Computation*, 88, 1990.

[45] Microsoft. *Visual Basic Programmer's Guide*, 1996.

[46] A. Middeldorp and M. Starcevic. A rewrite approach to polynomial ideal theory. Technical report, CWI Amsterdam, 1991.

[47] G. Milne and R. Milner. Concurrent processes and their syntax. *Journal of the ACM*, 1979.

[48] R. Milner. A Calculus of Communicating Systems. *Lectures Notes in Computer Science*, 92, 1980.

[49] R. Milner. *Concurrency and Communication*. Prentice Hall, 1989.

[50] R. Milner. Interaction. turing award lecture. *Communication of the ACM*, 1993.

[51] M. Mukherji and D. Kafura. CCE: A process calculus based formalism for specifying multi object coordination. *LNCS*, 1061, 1996.

[52] G.M.P. O'Hare and N. Jennings. *Foundations of Distributed Artificial Intelligence*. John Wiley, 1996.

[53] J.L. Peterson. *Petri Net Theory and the Modelling of Systems*. Prentice Hall, 1981.

[54] L. Robbiano. Introduzione all'algebra computazionale. Technical report, Dipartimento di Matematica. Univ di Genova, 1987.

[55] S. Stifner. A generalization of reduction rings. *Journal Symbolic Computation*, 4, 1987.

[56] M. Tokoro. The society of objects. In *Proc. of OOPLSA 94*, 1994.

[57] F. Winkler. Knuth Bendix procedure and Buchberger algorithm: a syntesis. In *ISAAC-89*, 1989.