Computational Models for Information Reuse

FRANCESCA ARCELLI FONTANA[1], FERRANTE FORMATO[1] AND REMO PARESCHI[2]

[1]*DIIIE, University of Salerno, I-84084 Fisciano (SA), Italy*
[2]*Telecom Italia, Divisione Ricerca e Sviluppo, Italy*
Email: arcelli@dia.unisa.it

Techniques that optimize computations by reusing partial results have a long tradition in computer science. Seen from the point of view of sequential computations, all these techniques share the common execution strategy of storing partial results in a centralized data structure, while they differ as to how the results are computed, e.g. in a data-driven or constraint-driven fashion. Concurrent systems, namely the wide variety of systems that range from fine-grained parallelism to coarse-grained distribution, add another variable into the game. In fact, information reuse is here tangled with issues of local memory of agents and inter-agent communication. Thus, optimal strategies for information reuse directly affect agent configurations as well as agent communication protocols and strictly depend on those morphological aspects of the computational domain related to the sharing of structures among different data values. In this paper, we define a formal framework suitable for the study of information reuse from the point of view of concurrent systems. The main result of our work is in the identification of two distinct morphological features of computational domains, namely *recursively replicated structures* and *structure copying*. These features induce two different forms of information reuse that can be optimized, respectively, by *solipsistic* agents with large local memory and by large bandwidth networks of *collaborative* agents.

Received April 21, 1998; revised September 14, 1999

1. INTRODUCTION

Information reuse techniques are involved at several levels and in different fields, both for sequential and concurrent computations, representing a central topic in computer science, receiving contributions from such diverse fields as mathematical programming [1], formal languages (see [2] and references cited therein) and query processing (see [3] for an overview). In particular if we consider concurrent systems, namely the wide variety of systems that range from fine-grained parallelism to coarse-grained distribution, information reuse is tangled with issues of local memory of agents and inter-agent communication, where a concurrent system is informally characterized as a set of agents and communication protocols. Thus, optimal strategies for information reuse directly affect agent configurations as well as agent communication protocols, and strictly depend on those morphological aspects of the computational domain related to the 'sharing of information'.

For an agents-based system we mean a system composed by a collection of agents, with their own memory, working in parallel and communicating in an organized way through different protocols. In particular our attention to concurrent-agent-based systems is related to the aspects involved by the communication protocols used by agents [4] and to their configuration in terms of their own memory. Our results can be applied in several fields and real-life applications. We give some hints in this direction throughout the paper and in the conclusions. Some domains in which we can exploit the techniques described are, for example, the area of distributed intelligent agents [5], distributed case-based reasoning [6] and cooperative information gathering [7]. It is out of the scope of our work, since it is not relevant for our analysis, to take into account the agent's features, such as reasoning capability, autonomy, intelligence, etc., largely discussed in the literature (see for example [8]).

In the paper, we define a formal framework suitable for the study of information reuse from the point of view of concurrent systems.

To give an idea of the problems that we tackle, consider the case of a system for distributed information retrieval [9], with agents selecting and merging information from multiple repositories in a wide-area network [10]. Here the computational domain is built from the data supplied by the information providers, and combined according to given query operators. It may happen that some items can be reused several times as answers to different queries. This is a typical case of 'structure sharing' in a computational domain. Queries define a 'scope of search', i.e. a constraint; computational agents have to generate all of the items in the computational domain that satisfy the query. Queries can be decomposed into sub-queries with newly spawned agents taking dynamic responsibility for each sub-query. Generally speaking, agent configurations in this application domain will tend to either one of the following three cases.

(i) Items in the computational domain are characterized by little structure sharing; this happens when items returned by a query agent do not contain sub-structures shared by other result items. This is typical when a query can be decomposed into a set of independent (i.e. disjoint) sub-queries. Caching of results may in this case be still useful, for instance for the computational support of long query sessions where the same item may be requested again and again in the context of different user queries. However, there is little room for devising optimal query strategies by setting up architectures which privilege local memory of agents with respect to bandwidth or vice versa, as neither storing nor communicating large amounts of results can be exploited in order to optimize the query performance.

(ii) There is a lot of structure sharing among items falling within the *same* scope of search. In this case, concurrent architectures privileging local storage against bandwidth will be optimal. In fact, each agent just needs to accumulate as many results as possible for the whole system to perform optimally.

(iii) There is a lot of structure sharing among items falling within *different* scopes of search. In this case, availability of bandwidth is very important in order to achieve good performance, while local memory is less relevant, as reuse is an inter-agent rather than an intra-agent fact.

The purpose of this paper is to define a formal framework to identify the conditions under which the morphology of a 'computational domain' tends either to (ii) or to (iii) in relation to the analysis of reusable structures. Such conditions will be, indeed, precisely characterized via the dual morphological features of *recursively replicated structures* and *structure copying*. For each of the two reusable structures, we can derive two communication protocols, which correspond to the cases of information reusability described in (ii) and (iii), respectively.

To this aim, we consider the *algebraic construction* and the *context-free generation* of a computational domain and we identify two particular computational models through which the computational domains can be generated. These computational models correspond to two kinds of distributed architectures: a *data-driven architecture* suitable for rule-based systems and a *constraint-driven architecture* suitable for systems based on constraint solving. The computational model underlying the former architecture, the data-driven model, can be seen as an information generator from a set of initial data of a given computational domain, while the model underlying the second architecture, the constraint-driven model, can be seen as an information system whose output is achieved by solving a set of constraints.

A computational domain can be generated in either of the two models by a corresponding concurrent system and we will outline the strong relation that exists between the reusable structures inherent to the computational domain with the performance of the concurrent system which generates them. Hence we characterize information reuse from the two viewpoints of 'dynamic' and 'static' reuse. Techniques will be defined in the paper to determine whether a computational domain tends to these two different cases of information reuse. Finally, we introduce the notion of measure of reuse to quantify the reusable structures in a computational domain, which can be used as a measure of the efficiency of the solution adopted to choose the best architecture.

As an example of the exploitation of a structure of reuse, as the recursively replicated structure, one can consider the case of a documents search and dynamic assembly of electronic documents from multiple sources of data according to the document mark-up language SGML. One can image a spider for the Web that collects a set of structured indexes representing a document by means of an SGML-based grammar of the kind: $\langle \text{DOCUMENT} \rangle \rightarrow \langle \text{HEAD} \rangle \langle \text{BODY} \rangle$, $\langle \text{BODY} \rangle \rightarrow \langle \text{ELEMENT} \rangle \langle \text{BODY} \rangle$. The spider is represented by an agent that is capable of delegating a set of agents to build a tree of indexes representing a document in the database. Two distributed architectures can be used according to the agent's storage capability: if the agent has a few local stores, it will create an agent to look for the 'head' of the document and an agent to build the 'body' of the document. In its turn, this latter agent will create two other agents, one to build the first element of the document and the other to build the rest of the body. This process will continue recursively, performed by collaborative agents with a high communication overhead. If the agent has a large local store, only two agents need to be created, because one agent, a solipsistic agent, is enough to build the body of the document by storing partial results.

Otherwise, as an example of the exploitation of the other structure of reuse, the structure copying, one can think of a query submitted to an agents-based system of the kind (A and B) or (A and C); this query can be formulated for example in this way: (Agents and Electronic_Commerce) or (Agents and Secure_Transactions). In this case, the structure copying-based protocol can be successfully used by collaborative agents with low storage capability.

The paper is organized through the following sections. Section 2 provides the formal background for the definition of computational domains and of reusable structures. Section 3 introduces the notion of *measure of reuse* to quantify reusable structures in a computational domain and provide a characterization of reuse within structures corresponding to items in that domain. Section 4 generalizes via asymptotic techniques the machinery developed in Section 3 to provide a characterization of reuse for full computational domains. Finally, Section 5 concludes and outlines some real-life applications of the reusable structures introduced in the paper.

2. DATA-DRIVEN AND CONSTRAINT-DRIVEN MODELS OF COMPUTATION

We consider here two different ways of modelling computations: *data-driven* models and *constraint-driven*

models. Data-driven models correspond to information generators that build new information from a (finite) set of initial data (the 'initial elements' of a domain) and a finite set of 'domain generators' (the 'constructors' of the domain). Informally speaking a generator is an object yielding new 'tokens of knowledge', i.e. elements of the computational domain, from existing ones. On the other hand, constraint-driven models can be seen as information systems that output information by propagating constraints over the elements of the domain using a set of rules associated with domain generators. Ultimately, the two approaches are equivalent: in fact one proves that any domain that is obtained by a set of generators can be specified as a set of constraints, and conversely, a domain that is specified by a set of constraints can be generated by a set of generators. According to the chosen approach, it is possible to study different computational techniques for the reuse of information, such as memorization [11], recursive query processing [3] and dynamic programming [1].

We now provide a formal definition of the notion of a computational domain.

2.1. Computational domains

Let F be a finite signature, i.e. a finite set of function symbols with an arity, and let C be a set of constants, i.e. a set of functions whose arity is zero, and $F \cap C = \emptyset$. Given a set X, we denote by $\mathcal{P}(X)$ the power set of X. We call *initial tokens* the elements of C and *domain generators* the elements of F.

2.1.1. Algebraic construction of computational domains

A computational domain can be defined through an algebraic operator, using a set C of initial tokens and a set F of domain generators.

DEFINITION 2.1. *Let $\mathcal{U}$ be a non-empty set. An* algebraic operator on $\mathcal{U}$ *is any map $T : \mathcal{P}(\mathcal{U}) \to \mathcal{P}(\mathcal{U})$ such that, for every $X, Y \in \mathcal{P}(\mathcal{U})$:*

(i) $X \subseteq T(X)$,
(ii) *if $X \subseteq Y$ then $T(X) \subseteq T(Y)$,*
(iii) *if $x \in T(X)$ then there exists a finite set $F \subseteq X$ such that $x \in T(F)$.*

Given an algebraic operator T, we define the powers of T as usual, i.e. we set $T^0 = i$ and $T^{i+1} = T \circ T^i$.

PROPOSITION 2.1. *Let T be an algebraic operator on $\mathcal{U}$. Then the following two properties hold:*

(i) *for any $X \in \mathcal{P}(\mathcal{U})$ there exists a fixed point M of T such that, if $Y \supseteq X$ and $T(Y) = Y$, then $Y \supseteq M$,*
(ii) $M = \bigcup_{n \in N} T^n(X)$.

We call M the *least fixed point of T that contains X*.

DEFINITION 2.2. *A computational domain D with a set of initial tokens C and a set of generators F is the least fixed point, that contains the empty set $\emptyset$, of the operator T on $\mathcal{P}(C \cup F)$ defined by setting, for any $X \subseteq C \cup F$:*

$$T(X) = \bigcup_{f \in F} \{f(d_1, \ldots, d_n) \mid d_1, \ldots, d_n \in X\}. \quad (1)$$

Hence

$$D = \bigcup_{n \in N} T^n(\emptyset).$$

Note that, if $F = \emptyset$, then $D = C$.

In other words, a computational domain could be incrementally constructed starting from a set of initial tokens C and applying the generators in F.

Given a signature F and a set of initial tokens C, we call the *algebra of closed terms on F and C* the free algebra over $F \cup C$ and we denote it with $M(F \cup C)$. Note that an algebra of closed terms over $F \cup C$ is also a computational domain.

2.1.2. Context-free grammars for the generation of computational domains

Another formalization of the computational domain can be achieved through context-free grammars. Let $G = \langle S, V_T, V_N, P \rangle$ be a context-free grammar, where S is the start symbol, V_T is the set of terminal symbols, V_N is the set of non-terminal symbols and $P = \{x_i \to w_i \alpha_i w_i', \ i = 1 \ldots n, w_i, w_i' \in (V_T \cup V_N)^*, \alpha_i, x_i \in V_N\}$ is the set of production rules. We denote by $L(G)$ the language generated by G.

We can associate an algebraic operator T_G to G by setting $T_G(X) = X \cup \{w\alpha w'$, where $x \to \alpha \in P, w, w' \in X\}$, for any $X \subseteq (V_T \cup V_N)^*$. It is easy to see that $L(G) = n \in N \bigcup T_G^n(\emptyset)$. As a consequence the set of context-free grammars is a particular class of computational domains.

We have considered two computational models, data-driven and constraint-driven models; a computational domain can be generated according to both of the two models. For example a string in a context-free language can be generated in a data-driven fashion, while a bottom-up query evaluation in a deductive database system is a typical example of a constraint-driven process.

In this paper, we show that the nature of the computational domain can strongly influence the performance of a concurrent system. In particular, there is a strong relation between the occurrence of reusable information in a computational domain and the performances of a concurrent system that generates it. For example, when a computational domain D presents the possibility of reusing partial results (e.g. a strongly recursive, deductive database), then in an agent-based concurrent system a communication protocol allowing reuse of partial results, together with large-memory agents, gives the best performance.

To provide a formal basis for the analysis of reuse of information in computational domains we make precise, in the next section, the notions of *recursively replicated structures* and of *structure copying*.

2.2. Reusable structures in a computational domain

Given a computational domain D, we call a *reusable structure* a general pattern that, when detected in an element

of D, can be used to improve the generation of this element in a concurrent system.

We can characterize information reuse from the two viewpoints of *static* and *dynamic* reuse. Static reuse is in a sense the amount of potential reuse intrinsic in a generator of D, i.e. static reuse is an *a priori* measure of reuse, that can be performed *independently* from the generation of any element of D. In contrast, dynamic reuse is the amount of reuse needed to generate any element in D, i.e. dynamic reuse is an *a posteriori* measure of reuse, *according* to the generation of a particular element of D.

Context-free grammars for the generation of computational domains are useful for studying information reuse from a static point of view. In fact, by viewing domain generators as context-free production rules, we shall be able to devise a method for determining *a priori* the *inherent reuse*, i.e. the quantity of reusable components, of the computational domain D generated by a corresponding context-free grammar.

On the other hand, by representing the computational domain D as an algebra of closed terms, we shall be able to study information reuse in D from the dynamic point of view. Indeed, by analysing reusable structures in the closed terms of the algebra, we will derive an asymptotic method for measuring reuse in D.

In particular, we have studied two kinds of reusable structures and we have found that, for each of these, a class of communication protocols could be defined to exploit such structures with the minimum amount of communication overhead. As shown in Section 4, this will allow one to 'measure the efficiency' of one solution with respect to the other.

We now characterize the two relevant cases of reusable structures: *recursively replicated structures* and *structure copying*. Later, in Section 3, we develop some techniques, based on the algebraic construction of computational domains, for detecting such reusable structures.

2.2.1. *Recursively replicated structures*

Consider a computational domain $M(F \cup C)$ where $F = \{f/1\}$, and $C = \{a\}$. Thus, all elements in the domain will be of the form $f^n(a)$, where $f^0(a) = a$ and $f^n(a) = f(f^{n-1}(a))$. This corresponds to a *recursively replicated structure* of the kind $Element = f(Element)$. As a consequence, in a concurrent agent-based computational system a *solipsistic* (non-communicating) agent can be allocated to generate all elements in $M(F \cup C)$ of length $\leq n$. The actual value of n will be determined by the size of the local memory of the agent, and in this case the optimal strategy that can be followed by the agent is caching $f^{n-1}(a)$ in its memory and then generating $f^n(a)$.

This kind of reusable structure is detectable in two main ways. In computational domains that are also free algebras of closed terms, it could be observed by simply analysing the structure of a term in the algebra. In a context-free language such kind of reuse can be detected through the analysis of the set of production rules of the corresponding grammar.

2.2.2. *Structure copying*

Another kind of reusable structure is given by *structure copying*, i.e. copies of the same structure repeatedly used by a generator of the computational domain. For instance, in the algebra $M(F \cup C)$ where $F = \{f/2, g/2\}$ and $C = \{a\}$; the term $f(g(a, a), g(a, a))$ is generated by two copies of the term $g(a, a)$. Such reuse is relatively simple to detect, both in free algebras and in context-free languages, as will be shown later.

Again, the detection of structure copying can be useful for defining optimal strategies for concurrent computation. More generally, structure copying is optimally supported by large bandwidth networks of *collaborative* agents, where the same item, once computed by a given agent, can be cheaply served to other agents that may need it for the execution of their own computations.

2.2.3. *Example: reusable structures in a deductive database*

We now give an example of query evaluation in logic programming [12] to illustrate the reuse of information related to recursively replicated structures and structure copying.

EXAMPLE 1. Let P be the following logic program:

$$p(f(x, y), f(z_1, z_2)) \leftarrow p(x, z_1), p(y, z_2). \quad P1$$
$$p(g(x, y), g(z_1, z_2)) \leftarrow p(x, z_1), p(y, z_2). \quad P2$$
$$p(h(x), h(z_1)) \leftarrow p(x, z_1). \quad P3$$
$$p(a, a). \quad P4$$

Consider the goal $G = p(f(f(X, Y), h(X)), Z)$.

The top-down evaluation (for example using SLD resolution) of the above goal can be performed in a concurrent system by taking advantage of the reuse derived by recursively replicated structures.

In fact, the sequence of resolvents is the following:

$$R_0 = G = p(f(f(X, Y), h(X)), Z)$$
$$R_1 = p(f(X, X), Z1), p(h(Z2), Z3)$$
$$R_2 = p(X, Z4), p(X, Z5), p(h(Z2), Z3)$$
$$R_3 = p(a, Z5), p(h(Z2), Z3)$$
$$R_4 = p(h(Z2), Z3)$$
$$R_5 = p(Z2, Z6)$$
$$R_6 = [].$$

Suppose that to each clause $P1$, $P2$, $P3$ and $P4$ in P is assigned an agent A1, A2, A3 and A4, respectively. Also, assume that each of the agents that manage rules can cache partial results in order to reuse them. The goal G is passed to agent A1 that delegates the subgoal $p(f(X, X), Z1)$ to itself and the subgoal $p(h(Z2), Z3)$ to A3.

Self-delegation is allowed because a recursively replicated structure of kind $f(f(....))$ occurs in the goal G. On receiving $p(f(X, X), Z1)$, A1 delegates execution of goal $p(X, Z4)$ to A4. A4 responds with $p(a, a)$, and it broadcasts this answer to the local cache of A1, A2 and A3. By so

doing, when A1 asks for $p(X, Z5)$, it can find the answer $p(a, a)$ in its local memory, without delegating the query to other agents and augmenting the communication overhead in the agent system. In the same manner, when A3 must solve the goal $p(h(Z2), Z3)$ it finds the answer directly in its local cache. This is allowed since in the goal G several copies of the same structure occur.

We want to outline that some methods of reusing information that exploit reusable structures, such as recursively replicated structures and structure copying, have been already introduced for query evaluation of logic programs. For example a very similar evaluation strategy, exploiting the storage of intermediate results and the reuse of them when needed, is the recursive version of the QSQ (query/subquery) [13] evaluation strategy. In this method, the top-down evaluation evolves through a set of tuples until no new tuple can be generated. When recursion is involved, only new tuples matter, since the old ones are already generated by recursion and reused. Therefore, recursive QSQ methods are able to detect cycles in the facts. QSQ evaluation can be seen as an early attempt of exploiting recursively replicated structures occurring in a computational domain.

3. MEASURES OF REUSE

In this section, we shall introduce the notion of *measure of reuse* to quantify reusable structures in a computational domain, viewed as an algebra of closed terms. This measure of reuse can be exploited to choose the best communication protocol between agents.

Given a set of generators F and a set of initial tokens C, the algebra of closed terms $M(F \cup C)$ is the least fixed point with respect to $\emptyset$ of the following algebraic operator on $F \cup C$:

$$T : X \in \mathcal{P}(F \cup C) \rightarrow X \cup \{f(t_1, \ldots, t_n), \text{where } t_1, \ldots, t_n \in X,\ f \in F\}. \quad (2)$$

Thus, reusable structures can be determined on the basis of the syntax of the closed term $t \in M(F \cup C)$.

3.1. A measure of reuse for propositional formulae

We apply our techniques to the domain of propositional formulae and then generalize them to any computational domain.

The set of propositional formulae, with set of atoms $\{a, b\}$, is a context-free language generated by the grammar $G = (\{F\}, \{F\}, \{a, b, \wedge, \vee, \neg\}, P)$ where P is defined as follows:

$$F \Rightarrow A|(F \wedge F)|(F \vee F)|(\neg F)$$
$$F \Rightarrow a|b.$$

Note that $L(G)$ can also be seen as an algebra of terms over $\{\vee, \wedge, \neg\} \cup \{a, b\}$. Let $\otimes$ be a variable taking values in $\{\vee, \wedge\}$. The measure of reuse for propositional formulae that quantifies the amount of structure copying in a formula, can be defined as follows.

For any formula α, the multiset $SF(\alpha)$ of its subformulae is defined as:

$$SF(\alpha) = \begin{cases} \{A\} & \text{if } \alpha \in Atom \\ SF(\gamma) \overset{\circ}{\cup} SF(\beta) & \text{if } \alpha = \gamma \otimes \beta \\ SF(\beta) & \text{if } \alpha = \neg\beta. \end{cases}$$

where $A \overset{\circ}{\cup} B$ is the union of two multisets A and B, i.e., for an element a, if $\mu_A(a)$ and $\mu_B(a)$ are the multiplicities of occurrences of a in A and B, respectively, then $\mu_{A \overset{\circ}{\cup} B}(a) = \mu_A(a) + \mu_B(a)$. The set of elements of $SF(\alpha)$ whose multiplicity is strictly greater than one is denoted with $\Delta(SF(\alpha))$.

DEFINITION 3.1. *Let α be a propositional formula, then the reuse measure related to structure copying of α, denoted by Re_{sc}, is defined by setting, for any formula α:*

$$Re_{sc}(\alpha) = \sum_{\beta \in \Delta(SF(\alpha))} \mu_{SF(\alpha)}(\beta).$$

The idea underlying the above definition is a sort of *reuse award* that a formula receives on account of the number of structures copied in its subformulae.

DEFINITION 3.2. *Let α and α' be two propositional formulae. We say that $\alpha \doteq \alpha'$ if $\alpha = (\gamma \otimes (\beta \otimes \delta))$ and $\alpha' = ((\gamma \otimes \beta) \otimes \delta)$. We denote with $\overset{*}{=}$ the equivalence relation generated by $\doteq$.*

We have the following theorem.

THEOREM 3.1. *Let α, α' be two formulae. Then, if $\alpha \overset{*}{=} \alpha'$ then*

$$Re_{sc}(\alpha) = Re_{sc}(\alpha'). \quad (3)$$

Proof. By a simple induction on the structural complexity of a formula:

(1) If α is an atom, then, trivially, $Re_{sc}(A) = 0,\ \forall A \in Atom$.
(2) If α is a non-atomic formula, we distinguish two different cases:
   (a) $\alpha = \beta \otimes \gamma$. If $\alpha' \overset{*}{=} \alpha$ then, by the theorem of unique decomposition of propositional formulae, it follows that $\alpha' = \beta' \otimes \gamma'$, where $\alpha \overset{*}{=} \alpha'$ as well as $\beta \overset{*}{=} \beta'$. Thus, if $\beta \overset{*}{=} \gamma$, then $Re_{sc}(\alpha) = Re_{sc}(\beta \otimes \beta) = 2Re_{sc}(\beta) + 1 = 2Re_{sc}(\beta') + 1$, by hypothesis of induction. Hence, by definition we get $2Re_{sc}(\beta') + 1 = Re_{sc}(\beta' \otimes \beta') = Re_{sc}(\beta' \otimes \gamma') = Re_{sc}(\alpha')$. The case where $\beta \neq \gamma$ is similar.
   (b) $\alpha = \neg\beta$. By the theorem of unique decomposition of propositional formulae, we get $\alpha \overset{*}{=} \alpha'$ iff $\alpha' = \neg\beta'$ where $\beta \overset{*}{=} \beta'$. Thus, $Re_{sc}(\neg\alpha) = Re_{sc}(\beta) = Re_{sc}(\beta')$ by definition of Re_{rs}. Hence, by the induction hypothesis, $Re_{sc}(\beta') = Re_{sc}(\neg\beta') = Re_{sc}(\alpha')$. □

We give here an example of the application of Definition 3.1.

EXAMPLE 2. Let $\alpha = (A \vee A) \vee (A \vee A)$. We have that $SF(\alpha) = \{A \vee A, A \vee A, A, A, A, A\}$. Therefore, $Re_{sc}((A \vee A) \vee (A \vee A)) = 6$ and hence the amount of reuse due to structure copying is six.

If we now set $\alpha_M(n)$ as follows:

$$\alpha_M(1) = A \quad \text{where } A \text{ is an atom}$$
$$\alpha_M(n) = \alpha_M(n-1) \otimes \alpha_M(n-1)$$

where $\otimes$ is a syntactic variable ranging in $\{\vee, \wedge\}$, it turns out that $\alpha_M(n)$ is the formula that maximizes the structure copying in $L(G)$, for a fixed number n of atomic occurrences. The formula $\alpha_M(n)$ contains 2^{n-1} occurrences of atomic formulae (this can be represented as a binary tree with 2^{n-1} leaves). Note that $Re_{sc}(\alpha_M(1)) = 0$ and $Re_{sc}(\alpha_M(n)) = 2 + 2Re_{sc}(\alpha_M(n-1))$ for any $n > 0$.

The length of a propositional formula is defined as: $length(A) = 1$ if A is an atom, and $length(\beta \otimes \gamma) = length(\beta) + length(\gamma)$ otherwise.

PROPOSITION 3.1. *For any propositional formula α of* length *n:*

$$Re_{sc}(\alpha) \leq Re_{sc}(\alpha_M(n)). \tag{4}$$

Proof. We proceed by induction on the number n of atomic occurrences.

(1) For $n = 1$, the proposition is trivially true.
(2) Let the proposition be true for n. For any propositional formula α of length $n+1$, we have that $\alpha = \beta \otimes \gamma$.

Hence, $Re_{sc}(\beta \otimes \gamma) \leq Re_{sc}(\beta) + Re_{sc}(\gamma) + 2 \leq Re_{sc}(\alpha_M(n)) + Re_{sc}(\alpha_M(n)) + 2 = 2Re_{sc}(\alpha_M(n)) + 2 = Re_{sc}(\alpha_M(n+1))$. □

We now define the measure of reuse quantifying the recursively replicated structures. In this case, 'reuse awards' are attributed to formulae characterized by recursive replication of structures. Indeed, the award is given if, in some subformula, there is a connective capable of reusing itself in a higher-level formula, as in $A \vee (A \vee A)$. Such an operand augments the measure of reuse, as expressed in the following definition.

DEFINITION 3.3. *Let α, β and γ be three well-formed formulae, and let* Atom *be the set of propositional atomic formulae. Furthermore, let $\otimes$ and $\oplus$ be two variables ranging in $\{\vee, \wedge\}$ assuming different values. The measure of reuse derived from recursively replicated structures, denoted with Re_{rs}, is defined as follows:*

$$Re_{rs}(\gamma) = \begin{cases} 0, & \text{if } \gamma \text{ is an atom} \\ Re_{rs}(\beta) & \text{if } \gamma = \neg\beta \\ Re_{rs}(\alpha \otimes \beta) + Re_{rs}(\delta) + 1 & \text{if } \gamma = \alpha \otimes \beta \otimes \delta \\ Re_{rs}(\alpha \otimes \beta) + Re_{rs}(\delta) & \text{if } \gamma = \alpha \otimes \beta \oplus \delta. \end{cases} \tag{5}$$

Informally speaking, the idea of 'recursively replicated structure' is to give a 'reuse award' to those formulae, such as $\alpha \wedge (\alpha \wedge \alpha)$, where the 'outer' connective is the same as the nested 'inner' connective.

Theorem 3.2 immediately follows.

THEOREM 3.2. *Given the propositional formulae α and α', if $\alpha \stackrel{*}{=} \alpha'$ then:*

$$Re_{rs}(\alpha) = Re_{rs}(\alpha'). \tag{6}$$

3.2. A generalized measure of reuse

In Section 3.1 we have shown that the reuse awards rely only on syntactic structures and do not depend on the semantic properties of the term structure. Thus, we can generalize the measure of reuse derived in Section 3.1 to *any* algebra of closed terms $M(F \cup C)$ on a set of generators F and a set of initial tokens C.

Let $M(F \cup C)$ be an algebra over $F \cup C$. For any term t in $M(F \cup C)$ we define the multiset $ST(t)$ of subterms of t occurring in t:

$$ST(t) = \begin{cases} \{t\} & \text{if } t \text{ is a constant} \\ \circ^n \bigcup_{i=1} ST(t_i) & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

For any term t, we define the set $\Delta(ST(t))$ as the multiset of elements of $ST(t)$ whose multiplicity is greater than one.

DEFINITION 3.4. *The generalized measure of reuse associated with structure copying, denoted with Re_{gsc}, is defined by setting, for any $t \in M(F \cup C)$:*

$$Re_{gsc}(t) = \sum_{t' \in \Delta(ST(t))} \mu_{ST(t)}(t').$$

The definition above is a generalization of Definition 3.1.

We now generalize the measure of reuse for recursively replicated structures. The main idea is that the 'reuse award' is given to a term of the kind $f(\ldots, f, \ldots)$, i.e. to a term where the symbol of a subterm-level function of F becomes a top-level operator.

The generalized measure of reuse related to recursively replicated structures should take into account the number of substructures that can be reused in the generation of the term.

Hence, we have:

DEFINITION 3.5. *Let $M(F \cup C)$ be an algebra over $F \cup C$. The generalized measure of reuse associated with recursively replicated structures, denoted with Re_{grs}, is given as follows:*

$$Re_{grs}(t) = \begin{cases} 0 & \text{if } t \text{ is a constant} \\ \sum_{i=1}^{n} m(t_i) \times Re_{grs}(t_i) + k & \text{if } n \geq 1,\ f \in F \\ & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

where $k = card(\{t_i, i \in \{1, \ldots, n\}$ such that $head(t_i) = f\})$, and where $m(t_i)$ denotes the multiplicity of occurrences of t_i in $f(t_1, \ldots, t_n)$.

We are now ready to give a more general definition of measure of reuse, by simply summing up the two components, Re_{gsc} and Re_{grs}.

DEFINITION 3.6. *Given an algebra $M(F \cup C)$ we define the measure of reuse on $M(F \cup C)$ the mapping $Re : M(F \cup C) \rightarrow \mathcal{N}$ such that, for any $t \in M(F \cup C)$: $Re(t) = Re_{gsc}(t) + Re_{grs}(t)$.*

We now show through an example, an application of the generalized measure of reuse, highlighting the differences between the two kinds of reuse (structure copying and recursively replicated structures).

EXAMPLE 3. Let $F = \{g/3, f/2\}$, $C = \{a/0, b/0\}$, and let $g(f(a,a), f(a,a), f(a,a))$ be a term in $M(F \cup C)$. The reuse amount due to structure copying is

$$ST(t) = \{f(a,a), f(a,a), f(a,a), a, a, a, a, a, a\}$$

and hence

$$Re_{gsc}(g(f(a,a), f(a,a), f(a,a))) = 3 + 6 = 9.$$

The reuse amount due to recursively replicated structures is

$$\begin{aligned} &Re_{grs}(g(f(a,a), f(a,a), f(a,a))) \\ &\quad = 3 \times Re_{grs}(f(a,a)) + 0 \\ &\quad = 3 \times 0 + 0 = 0. \end{aligned}$$

This means that there is no recursively replicated structure in the term and so

$$\begin{aligned} &Re(g(f(a,a), f(a,a), f(a,a))) \\ &\quad = Re_{gsc}(g(f(a,a), f(a,a), f(a,a))) = 9. \end{aligned}$$

Using these measures of reuse, it is possible to give a characterization of the global amount of reuse inherent to a certain algebra of closed terms $M(F \cup C)$ (i.e. the computational domain $M(F \cup C)$). Essentially, we simplify the algebra $M(F)$ by considering an equivalence relation $\cong$ on $M(F)$ such that, if $t \cong t'$, then $Re(t) = Re(t')$. We repeat this process and we get a computational domain D^* that is equivalent to $M(F)$ from the point of view of the amount of reusable structures, whose cardinality is smaller than that of the initial domain $M(F)$.

3.3. Capturing hidden replicated structures

The general definition of the measure of reuse given in Definition 3.6, although useful in most cases, is not suitable to capture 'replicated structures' of terms hidden by unary operators. Indeed, unary operators are 'transparent' to the replication of structures and may hide subterms that can be effectively reused. To clarify the problem, consider again the algebra $M(F)$ of propositional formulae (as defined in Section 3.1), and consider the formula $\alpha \vee (\neg\alpha)$. According to Definition 3.6, $Re_{sc}(A \vee (\neg A))$ is zero. However, if we take the formula $A \vee A$, $Re_{sc}(A \vee A) = 2$. This happens since in $A \vee (\neg A)$ there is a replicated structure, namely the atom A, which is hidden by the unary operator $\neg$. In this sense, $A \vee (\neg A)$ and $A \vee A$ are equivalent.

Two approaches can be followed to resolve this problem.

(i) Making the measure of Definition 3.6 independent from the occurrence of the unary operators.
(ii) Considering the quotient of the algebra of closed terms with respect to an equivalence relation which identifies two terms up to the occurrences of unary operators.

We follow the latter approach and, given a free algebra $M(F \cup C)$ on a set of generators F and a set of initial tokens C, we define a binary relation $\sim_{re}$ on $M(F \cup C)$ as follows: $t \sim_{re} t'$ iff $t' = f(t)$ for some unary function $f \in F$. Given an algebra of terms $M(F \cup C)$, a *congruence* on $M(F \cup C)$ is an equivalence relation $\simeq$ on $M(F \cup C)$ such that, if $t_i \simeq t_i'$ then $f(t_1, \ldots, t_n) \simeq f(t_1', \ldots, t_n')$ for any $f \in F$.

DEFINITION 3.7. *Let $M(F \cup C)$ be an algebra of closed terms. The* reuse algebra *associated with $M(F \cup C)$, denoted with $M(F \cup C)_{re}$, is defined as the quotient of $M(F \cup C)$ with respect to the congruence $\simeq_{re}$ generated by $\sim_{re}$.*

The basic idea is the elimination of all unary operators from a term $t \in M(F \cup C)$. In the following, we shall define a measure of reuse over $M(F \cup C)_{re}$. To this purpose we introduce the notion of *reuse normal form.*

DEFINITION 3.8. *Let $M(F \cup C)$ be an algebra of closed terms. For any $t \in M(F \cup C)$, the reuse normal form $nf_{re}(t)$ is defined by structural induction over t:*

$$nf_{re}(t) = \begin{cases} 0 & \text{if } t \in C \\ nf_{re}(t') & \text{if } t = f(t'), t' \in M(F \cup C) \\ f(nf_{re}(t_1), \ldots, nf_{re}(t_n)) & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

In other words, the reuse normal form $nf_{re}(t)$ is obtained by eliminating the occurrences of unary operators in the term t.

The following lemma relates an algebra $M(F \cup C)$ with its associated reuse algebra $M(F \cup C)_{re}$.

LEMMA 3.1. *Let $M(F \cup C)$ be an algebra of closed terms, $t, t' \in M(F \cup C)$ and $F^+ = F - \{f \in F \mid f$ is a unary operator$\}$. Then the following two properties hold:*

(i) *for any $t \in M(F \cup C)$ $nf_{re}(t) \simeq_{re} t$,*
(ii) *for any $t, t' \in M(F \cup C)$, if $t \simeq_{re} t'$ then $t = t'$.*

Proof. (i) We proceed by the induction hypothesis on the structural complexity of terms: if $t \in C$ then $nf_{re}(t) = t$. If $t = f(t_1, \ldots, t_n)$, then $nf_{re}(f(t_1, \ldots, t_n)) = f(nf_{re}(t_1), \ldots, nf_{re}(t_n)) \simeq f(t_1, \ldots, t_n)$.
(ii) can be obtained by structural induction over the term t. □

The following proposition states that the notion of reuse normal form is invariant with respect to the relation $\simeq_{re}$.

PROPOSITION 3.2. *Let $t, t' \in M(F \cup C)$. If $t \simeq_{re} t'$ then $nf_{re}(t) = nf_{re}(t')$.*

Proof. The immediate consequence of Lemma 3.1. □

We are now ready to define the measure of reuse for an algebra of closed terms $M(F \cup C)$ with respect to its associated reuse algebra $M(F \cup C)_{re}$.

DEFINITION 3.9. *Let $M(F \cup C)$ be an algebra of closed terms. Let $M(F \cup C)_{re}$ be its associated reuse algebra. The measure of reuse in $M(F \cup C)_{re}$ is the mapping $Re^* : M(F \cup C)_{re} \to \mathcal{N}$, defined by setting, for any equivalence class $[t] \in M(F \cup C)_{re}$:*

$$Re^*([t]) = Re(nf_{re}(t)). \tag{7}$$

Proposition 3.2 assures that this definition is well posed and solves the problem of the hidden replication of structures. For instance, in the algebra $M(F_0)$ of propositional formulae, since $(A \vee (\neg A)) \simeq_{re} (A \vee A)$, we have that, in $M(F_0)_{re}$, $Re^*((A \vee (\neg A))) = Re(nf_{re}((A \vee (\neg A)))) = Re((A \vee A)) = 2$. This gives the exact amount of inherent reuse.

4. ANALYTICAL STUDY OF INHERENT REUSE

The reuse algebras introduced in Section 3.3 are a first step towards the analytical study of the structures of reuse in a computational model. In this section a computational domain is either a context-free grammar or an algebra of closed terms. In both cases, the notion of length of a token is given. Let M be a subset of a computational domain D. We denote by M^n the set of tokens in M of length n. The basic idea underlying our analytical approach to the inherent reuse in a computational model is based on the asymptotic behaviour of the quantity:

$$\lambda_{rs}^n = 1 - \frac{card(\hat{D}^n)}{card(D^n)} \tag{8}$$

where $\hat{D}$ is the set of tokens in D whose generation does not benefit from the reuse associated with recursively replicated structures. The asymptotic behaviour of the quantity λ_{rs}^n is dependent on the limit of the succession λ_{rs}^n. When this is close to one, then the computational domain will have an inherent structure of reuse which is prevailingly derived from recursively replicated structures. In contrast, when λ_{rs}^n tends to zero the other structures of reuse, derived from structure copying, will prevail. At this point, the problem of the complexity of D and, consequently, of $\hat{D}$ arises. By 'complexity problem', we mean here, intuitively, the need of defining a sound method that allows simplifying, as much as possible, a given computational domain, keeping invariant the information related to reuse of partial results in a distributed architecture that computes D. One way to tackle the complexity problem is to simplify the set D through the introduction of an equivalence relation that does not alter the information about the reusable structures in D. Therefore, a succession of equivalence relations $\simeq_i$, and a succession of computational domains D_i can be determined, such that, at each step i, the reusable structures in the domain D_i are kept unchanged. Finally, we determine a simplified computational domain D_k, such that the computation of $card(D_k^n)$ and $card(\hat{D}_k^n)$ is easier. Evidently, the value λ_{rs}^k relative to D_k gives the amount of reuse in D connected to recursively replicated structures. We apply this method to the set of propositional formulae $L(G)$ (see Section 3.1) and we define a sequence of equivalence relations that simplifies $L(G)$, allowing the analytical computation of the corresponding index λ_{rs}^k.

To this aim, we define a binary relation $\sim_1$ on $L(G)$ as follows: $\neg(\neg A) \sim_1 A$, $\neg(\neg\alpha) \sim_1 \alpha$, where A is an atom and α is a propositional formula.

Let $L(G)_1 = L(G)/\simeq_1$ where $\simeq_1$ is the equivalence relation generated by $\sim_1$. Let $L(G)_1^n$ be the set of strings in $L(G)_1$ whose length is n. We denote with C_n the nth number of Catalan, i.e. $C_n = (2n)!/[n!(n+1)!]$ and with Φ the cardinality of the set of atomic formulae in $L(G)$. The following proposition holds.

PROPOSITION 4.1. *For any integer n,*

$$|L(G)_1^n| = C_{n-1}\Phi^n 2^{3n-2}.$$

Proof. A bijection can easily be established between the binary trees, and the atomic formulae. For any tree and for any fixed combination of atomic formulae there are $2^{n-1} + 2^{2n-1}$ formulae of length n: indeed, the number of nodes that are not leaves, in a tree with n leaves, is $n-1$. Since every node has two leaves, and since the set of binary connectives is $\{\wedge, \vee\}$ there are 2^{n-1} ways of placing these connectives in a given binary tree. Moreover, for a given binary tree, there are

$$\sum_{i=1}^{2n-1} \binom{2n-1}{i} = 2^{2n-1}$$

ways of adding the negation symbol. Given a binary tree, there is no other formula that can be generated out of these two ways, since there are C_{n-1} different binary trees. The hypothesis thesis follows immediately. □

We now introduce an equivalence relation on $L(G)_1$.

DEFINITION 4.1. *We define the relation $\simeq_2$ on $L(G)_1$ as the smallest equivalence relation such that:*

$$\alpha = \beta \text{ or } \alpha = \neg\beta \Rightarrow \alpha \simeq_2 \beta$$
$$\alpha' \simeq_2 \beta' \text{ and } \alpha'' \simeq_2 \beta'' \Rightarrow \alpha' \wedge \alpha'' \simeq_2 \beta' \wedge \beta''$$
$$\alpha' \simeq_2 \beta' \text{ and } \alpha'' \simeq_2 \beta'' \Rightarrow \alpha' \vee \alpha'' \simeq_2 \beta' \vee \beta''.$$

Let $L(G)_2 = L(G)_1/\simeq_2$. We define a function $N : L(G) \to L(G)$ as follows:

$$N(\alpha) = \begin{cases} a & \text{if } a \text{ is an atom} \\ N(\alpha') \wedge N(\alpha'') & \text{if } \alpha = \alpha' \wedge \alpha'' \\ N(\alpha') \vee N(\alpha'') & \text{if } \alpha = \alpha' \vee \alpha'' \\ N(\alpha') & \text{if } \alpha = \neg\alpha'. \end{cases}$$

Intuitively, the function N takes a propositional formula α as input and gives as output the formula α without the occurrences of connective $\neg$. The following proposition is an easy induction on the structural complexity of propositional formulae in $L(G)$.

PROPOSITION 4.2. *Let α and β be two propositional formulae in $L(G)_1$. Then $\alpha \simeq_2 \beta$ iff $N(\alpha) = N(\beta)$.*

Proof. $\Rightarrow$) Let $\alpha \simeq_2 \beta$. Let $con(\alpha)$ be the number of connectives occurring in α. We proceed by induction on $con(\alpha) + con(\beta)$.

- If $con(\alpha) + con(\beta) = 0$ then α and β are atoms. As a consequence, $\alpha = \beta$ and therefore $N(\alpha) = N(\beta)$.
- If the claim is true for $0 < con(\alpha) + con(\beta) < n$ we prove that the claim is true if $con(\alpha) + con(\beta) = n$. Suppose that $\alpha = \alpha' \vee \alpha''$ and $\beta = \beta' \vee \beta''$. By definition, $\alpha' \simeq_2 \beta'$ and $\alpha'' \simeq_2 \beta''$. Therefore, by the induction hypothesis:

$$N(\alpha) = N(\alpha' \vee \alpha'') = N(\alpha') \vee N(\alpha'') = N(\beta') \vee N(\beta'') = N(\beta).$$

Suppose that $\alpha = (\neg\gamma)$. As a consequence, $\beta = (\neg\gamma')$. Therefore, by the induction hypothesis:

$$N(\alpha) = N(\gamma) = N(\gamma') = N(\neg\gamma') = N(\beta).$$

$\Leftarrow$) Analogous to the above proof. □

As a consequence, we can define a measure of reuse Re_2 on $L(G)_2$ by setting, for any equivalence class $[\alpha]_{\simeq_2} \in L(G)_2 : Re_2\{[\alpha]\} = Re_2\{N(\alpha)\}$.

PROPOSITION 4.3. *Let α and β be two propositional formulae. Then the following properties hold:*

(i) $\alpha \simeq_2 \beta \Rightarrow Re_2\{[\alpha]\} = Re_2\{[\beta]\}$
(ii) $card(L(G)_2^n) = C_{n-1}\Phi^n 2^{n-1}$.

Proof. (i) is an immediate consequence of the proposition above.

(ii) follows from the fact that, by representing a propositional formula with a binary tree, for any formula α of length n, there are 2^{n-1} ways of putting negations into it. □

We now determine the class of formulae whose parsing does not benefit from the consequences of recursively replicated structures. To this purpose, we define by induction the following set *Nors*, for any $\alpha, \alpha', \alpha'', \beta, \beta', \beta'', \in L(G)_2$:

$$\begin{aligned} Nors = Atom \cup \{&\alpha \oplus \beta \mid \alpha = \alpha'' \otimes \alpha', \\ &\beta = \beta' \otimes \beta'', \beta', \beta'', \alpha', \alpha'' \in Nors\} \\ \cup \{&\alpha \otimes \beta \mid \alpha = \alpha'' \oplus \alpha', \\ &\beta = \beta' \oplus \beta'', \alpha', \alpha'' \in Nors\}. \end{aligned}$$

Clearly, for any formula α in *Nors*, $Re_{rs}(\alpha) = 0$ and $card(Nors^n) = C_{n-1}\Phi^n$.

We can finally state the following theorem, whose proof is a consequence of what has been stated above.

THEOREM 4.1. *Let $L(G)_n$ be the set of propositional well-formed formulae whose length is n. Then:*

$$\frac{card(Nors^n)}{card(L(G)^n)} = \frac{1}{2^{n-2}}. \tag{9}$$

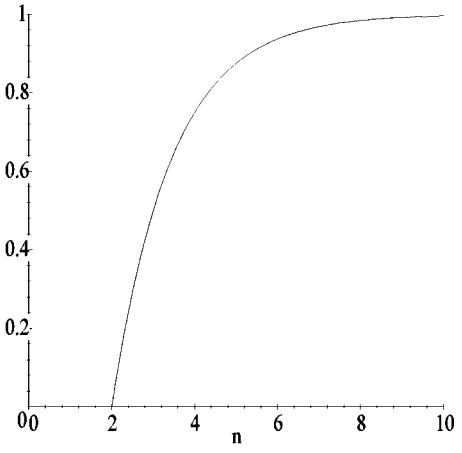

FIGURE 1.

As a consequence, in the case of propositional formulae we have that:

$$\lambda_{rs}^n = 1 - \frac{1}{2^{n-2}}.$$

Figure 1 is a graphic representation of λ_{rs}^n as a function of the length n of propositional formulae. It shows that, as n increases, the percentage of formulae that present recursively replicated reusable structures grows exponentially. This means that in the computational domain of propositional formulae, the best configuration of an agent-based system is achieved by solipsistic agents with large-memory capabilities.

5. CONCLUSIONS AND FUTURE DEVELOPMENTS

We have analysed information reuse from the point of view of concurrent systems in terms of the dual notions of *recursively replicated structures* and *structure copying*. These two alternative morphologies of computational domains determine different strategies and architectures for optimizing information reuse in concurrent systems. Indeed, solipsistic agents with large local memory serve the former case well, while the second one is ideally supported by collaborative agents connected via large bandwidth networks.

Related to the analytical study of inherent reuse, another direction of investigation can be pursued, when the computational domain is an unambiguous context-free language, and in this case the Chomsky–Schützenberger techniques [14] can be used as a uniform method to compute the cardinalities of $\hat{D}_n$ and D_n(defined in Section 4). We are studying some conditions, under which the process of quotientation of a context-free language can be *re-iterated* as many times as possible, in the attempt to reach a simpler computational domain D_n whose structures of reuse are exactly the same as the structures of reuse of the original domain D.

We observe that a general drawback of the algebraic approach is the lack of a scalability feature to suit complex

systems; nonetheless some algebraic methods, more linked to combinatorics, appear to be much more worthy. In fact, a complex system can be easily studied by means of a simplified version of it, obtained by quotienting the system with an equivalence relation that discards the useless information. Moreover, to compute the measures we have proposed, we need a method to calculate the cardinality of the computational domain involved. This is difficult in general, but there are several combinatorial tools [14] that allow these calculations in a large class of significant cases.

Presented in a less general form, some of the techniques in this paper have been applied [4, 15] to a constraint-based formalization of distributed information retrieval [16]. The next step is coupling them with an explicit formalization of concurrent systems. Among the many models of concurrency that have been proposed, frameworks based on graph rewriting [17, 18] appear as particularly appealing, since they make a clear (graphically representable) separation between local evolution of a single process (agent) and global synchronization of agents via interaction through communication ports. In this framework, it should be easy to represent and contrast solipsistic behaviour based on local computations and collaborative behaviour based on inter-agent communication.

We conclude by outlining some real-life applications. The protocols derived by structure copying and recursively replicated structures can be used for information propagation, and find applications in the domain of Internet- and intranet-based information filtering. Indeed, recursively replicated structure-based protocols can be used as a generalization of approaches to information filtering based on 'content push' that have become popular through several World-Wide Web applications. These applications enable the user to use the computer screen as a television with customizable information channels, where contents are selectively broadcast to users on the basis of content preferences that they have themselves specified. On the other side, the structure-copying-based protocol comes closer to 'collaborative filtering' approaches, where information selected as relevant by a user can be transmitted to, and reused by, users with similar interests.

REFERENCES

[1] Bellman, R. E. and Dreyfus, S. E. (1962) *Applied Dynamic Programming.* Princeton University Press, Princeton, NJ.

[2] Harrison, M. A. (1978) *Introduction to Formal Language Theory.* Addison-Wesley, Reading, MA.

[3] Bancilhon, F., Maier, D., Sagiv Y. and Ullman, J. (1986) Magic sets and other strange ways to implement logic programs. In *Proc. 5th ACM SIGACT/SIGMOD Symp. on Principles of Database Systems*, Cambridge, pp. 1–15. ACM Press, New York.

[4] Arcelli, F., Borghoff, U. M., Formato, F. and Pareschi, R. (1997) Constraint-based protocols for distributed problem solving. *Sci. Computer Programming*, **29**, 201–225.

[5] Sycara, K. *et al.* (1996) Distributed intelligent agents. *IEEE Expert*, **11**, 36–46.

[6] Durfee, E. H., Lesser, V. R. and Corkill, D. D. (1989) Cooperative distributed problem solving. In Barr, A., Cohen, P. R. and Feigenbaum, E. A. (eds), *The Handbook of Artificial Intelligence*, Vol. 4, pp. 83–148. Addison-Wesley, Reading, MA.

[7] Oates, T., Nagendra Prassad, M. V. and Lesser, V. R. (1994) *Cooperative Information Gathering: a Distributed Problem Solving Approach*. Technical Report TR-94-66, Department of Computer Science, University of Massachusetts, Amherest, MA.

[8] Wooldridge, M. and Jenning, N. R. (1995) Intelligent agents: theory and practice. *Knowledge Engng Rev.*, **10**, 115–152.

[9] Borghoff, U. M., *et al.* (1996). Constraint-based information gathering for a network publication system. In *Proc. 1st Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, London, UK, pp. 45–59. Blackpool, London.

[10] Andreoli, J.-M., Borghoff, U. M., Pareschi, R. and Schlichter, J. H. (1995) Constraint agents for the information age. *JUCS-electronic J.*, **1**, 762–789, also on http://www.iicm.edu/jucs.

[11] Hall, M. and McName, J. P. (1997) Improving software performance with automatic memoization. *J. Hopkins APL Tech. Digest*, **18**, 254–260.

[12] Das, S. K. (1992) *Deductive Databases and Logic Programming*. Addison-Wesley, Reading, MA.

[13] Vielle, L. (1986) Recursive axioms in deductive databases: the query–subquery approach. In *Proc. 1st Conf. on Expert Database Systems*, Menlo Park, CA. Benjamin/Cummings.

[14] Schützenberger, M. P. (1997) Sur une des Fonctions Sequentielles. *Theor. Comput. Sci.*, **4**, 47–57.

[15] Arcelli, F., Formato, F. and Pareschi, R. (1998) A measure of information reuse to compare distributed protocols. *Comput. Commun.*, **21**, 924–929.

[16] Andreoli, J.-M., Borghoff, U. M. and Pareschi, R. (1996) The constraint-based knowledge broker model: semantics, implementation and analysis. *J. Symbolic Computat.*, **21**, 635–667.

[17] Degano, P. and Montanari, U. (1987) A model for distributed systems based on graph rewriting. *J. ACM*, **34**, 411–449.

[18] Montanari, U. and Rossi, F. (1996) Graph rewriting and constraint solving for modelling distributed systems with synchronization. In *Proc. Int. Conf. on Coordination*, Cesena, Italy, Vol. 4.