Generalization of the r-operators

Bertrand Ducourthial
Laboratoire de Recherche en Informatique
email: ducourth@@lri.fr

June 3, 1998

Abstract:

This work deals with the termination of global computations performed in a distributed network of computing entities defined as follows. Each computing entity computes with a given operator a single value using those it receives from its direct ancestors in the network. This value is then sent to its direct descendants. The network is supplied at the beginning of the global computation with private datum of each entity. The system runs in asynchronous mode. Binary operators which are associative, commutative and idempotent (s-operators) allow to the global computation to terminate (Tel, 1994). In previous work, we introduced a more general class of operators, called the binary r-operators, that still allow termination of global computations.

In this report, we complete the definition of the associative nets in such a way that n-ary operators can be used by the entities. Then, we extend previous results to the n-ary r-operators. Next, we study some more general operators and we discuss about properties of operators which are necessary to insure termination of global computations. Finally, some applications of these operators such as distance computation, shortest-path-spanning-tree, depth-first-search, diameter computation and mathematical morphology are given.

Keywords: associative nets, r-operator, wave algorithm, distributed computation, distance computation, single-source-shortest-path, shortest-path spanning tree, depth-first-search, mathematical morphology

Résumé :

Ce travail concerne la terminaison de calcul globaux exécutés dans un réseau d'entités calculantes définies comme suit. Chaque entité calcule, avec un opérateur donné, une seule valeur à partir de celles qu'elle reçoit de ses antécédantes directes dans le réseau. Cette valeur est alors envoyée à ses descendantes directes dans le réseau. Au départ, le réseau est alimenté par les valeurs privées des entités. Le système fonctionne en mode asynchrone. Les opérateurs binaires qui sont associatifs, commutatifs et idempotents (s-opérateurs) assurent la terminaison du calcul global (Tel, 1994). Dans un précédent travail, nous avons introduit une classe plus générale d'opérateurs, appelés r-opérateurs binaires, et qui assurent encore la terminaison des calculs globaux.

Dans ce rapport nous complétons la définition des réseaux associatifs de telle manière que les entités puissent utiliser des opérateurs n-aires. Nous étendons alors les résultats précédents aux r-opérateurs n-aires. Ensuite, nous étudions des opérateurs plus généraux et nous discutons des propriétés des opérateurs nécessaires pour assurer la terminaison des calculs globaux. Finallement, des applications de ces opérateurs telles que le calcul de distance, l'arbre des plus courts chemins, l'arbre de poids minimum, l'arbre en profondeur, le calcul du diamètre et la morphologie mathématique sont données.

Mots clés: réseaux associatifs, r-operateur, algorithme en vague, calcul distribué, calcul de distance, parcours minimaux dans les graphes, arbre en profondeur, morphologie mathématique.

1 Introduction

In this report, we deal with distributed computing on a simple network of computing entities. In this Section, we briefly present the model and explain its use for parallel and distributed programming. Then we summarize related works, and we explain our aims.

The model A *computing entity* is the smallest unit of hardware or software that performs computations. Computing entities are linked together with directed links. The *network* is the set of inter-entities links. It works in asynchronous mode. A computing entity performs a *local computation* as soon as it receives a new datum. A local computation on the computing entity v is defined using a $\diamond_v$ operator. It consists in combining the datum of the entity with the received data using the $\diamond_v$ given operator. If the new result is different from the previous one, it is sent to the out-neighbor entities.

In this report, we allow non-uniform computations, that is, two computing entities u and v may not use the same operator ($\diamond_u \neq \diamond_v$).

A *global computation* begins when the network receives a signal from the outside. It consists of concurrent executions of the local computations on the network. It finishes when computations do not affect data in the entities any more (we say the global computation terminates). As in previous work, we call such a global computation a *direct–association*. Moreover, if each computing entity v uses an operator $\diamond_v$, the global computation is denoted by $\diamond_G$–*direct–association*, where $G(V, E)$ is the digraph that models the associative net. Hence a $\diamond_G -$*direct*$-$*association* is a global computation performed by concurrent executions of local computations based on the operators $\diamond_v$. The result of such a primitive exists only if the global computation terminates on a deterministic result. This computational model is called *Associative Nets*.

Associative nets The associative nets model is quite simple. It allows powerful implementations in hardware or software *(Mérigot, 1992)*. It is a general purpose computational model with high level programming primitives for computing (*direct*$-$*association*s) and for network management (not presented here) *(Ducourthial, 1995; Mérigot, 1997)*.

Associative nets have been used with binary operators which are commutative, associative and idempotent (we call such an operator an s-operator). With s-operators, the *direct*$-$*association*s always terminate in any network initialized with any data. These operators give useful programming primitives. For example, to broadcast a datum from a computing entity v over the whole network G, it is sufficient to set to zero all data except in v and then to perform a **max**$-$*direct*$-$*association* on G.

Associative nets have also been used with any kind of operators on directed acyclic graphs, and especially on leaves-to-root oriented trees. Since there is no cycle in these digraphs, all operators can be used and for example it is possible to perform a **plus**$-$*direct*$-$*association* on integers. This allows to compute the sum of the integers in any strongly connected digraph. Indeed, first a leaves-to-root rooted spanning tree is built. Then a **plus**$-$*direct*$-$*association* is performed. The sum of all integers of the network is obtained in the root entity. To broadcast the result of the sum over all the computing entities, a $\mathbf{max}-\mathit{direct}-\mathit{association}$ is used, as explained above.

In *(Ducourthial, 1997)*, we proposed to extend the use of associative nets to idempotent r-operators (which include the s-operators): we showed that any $\mathit{direct}-\mathit{association}$ based on an idempotent r-operator terminates in any network and with any data of the definition set A. The r-operators have interesting properties. In particular, we showed that any r-operator is based on an s-operator $\star$ and an isomorphism of $(A, \star)$. As an application, we presented the **minc** operator defined by $\mathbf{minc}(x, y) = \mathbf{min}(x, y + 1)$ and we showed this operator allows to compute distances in network in a single $\mathit{direct}-\mathit{association}$.

Related models A $\mathit{direct}-\mathit{association}$ can be compared to a *diffusing computation*, which is defined as follows *(Dijkstra and Scholten, 1980)*. The global computation starts when a computing entity receives an external signal from the environment. A computing entity can send messages to its neighbors when it receives an external signal or when it receives a message from one of its neighbor.

Some comparisons can be made with *wave algorithms*. An *half-wave* starts from a computing entity that sends messages to all its neighbors in the networks. At the first reception of a message, any entity resends it to its neighbors. This protocol is called *Propagation of Information (PI)* and has been proposed in *(Segall, 1983)*. A wave has been defined as a *Propagation of Information with Feedback (PIF)* where the feedback is defined as follows. At the end of the PI, the leaves of the diffusion-tree send messages to their fathers. Any intermediate node waits for a message from all its children. Next it sends a message to its own father. The root of the diffusion-tree (*i.e.* the entity that began the PI) knows that the protocol is ended when it has received a message from all its neighbors. This protocole has also been studied in *(Chang, 1982)* under the name of *echo algorithms*. In *(Hélary et al., 1987)*, an improvement in term of number of messages of these protocols is proposed. In *(Raynal and Hélary, 1988)*, they are used to compute a global function, on several topologies and with several communication hypotheses. In *(Tel, 1987)*, the PI and PIF protocols have been extended to strongly connected directed networks. In *(Tel, 1994)*, the concept of wave algorithm (called total algorithms in *(Tel, 1991)*) is generalized to the computation of an infimum (that we call s-operator in our work) over all the data of the network. A wave algorithm is then defined as a global computation which terminates in finite time and such that there is at least one result (decide event) which needs the participation of all computing entities. Wave algorithms can be used with any s-operator like $\wedge$, $\vee$, **min**, **max**, **gcd**, **lcm**, $\cap$ and $\cup$ *(Tel, 1994)* and also with idempotent r-operators *(Ducourthial, 1997)*.

A $\mathit{direct}-\mathit{association}$, when it terminates, is a wave algorithm where each computing entity has to build a result from data of all its ancestors (note that this corresponds to all the computing entities on an undirected network). Wave algorithms have many applications in distributed computations. A wave algorithm can be used to broadcast data, as we explained above with a $\mathbf{max}-\mathit{direct}-\mathit{association}$. This technique for broadcast is similar to the PI protocole. When the binary operator is not idempotent, the global computation needs a leaves-to-root rooted spanning tree, as explained above for the $\mathbf{plus}-\mathit{direct}-\mathit{association}$. This technique is similar to PIF protocol or echo algorithms.

Aims In this report, we define associative nets in a more general way, allowing the use of n-ary r-operators. Next we generalize results about r-operators and their ability to insure termination of global computations.

We explained above that a $\star$−*direct*−*association* terminates on a deterministic result if $\star$ is an s-operator. In *(Ducourthial, 1997)*, we add a transformation on all the received data but the first. Indeed, each incoming datum x is included in computation performed by the s-operator under the $r(x)$ form, where r is a mapping called r-mapping (Figure 1). When some conditions on the r-mapping are respected, the operator used by the computing entities is an r-operator.

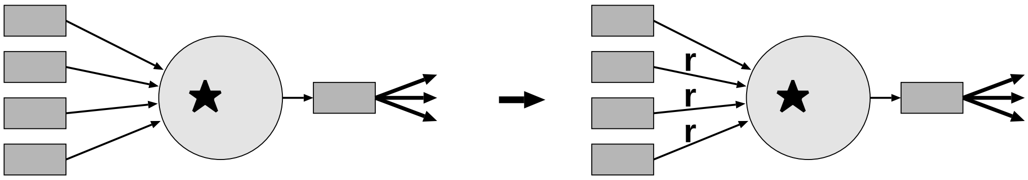

Figure 1: Extension of the s-operators to the binary r-operators.

In this report, we study some more general transformations on the incoming data before the computation performed by the s-operator. In the following, A is the set of the admitted values for computations. We call *rs-mapping* a mapping $\diamond_v$ from $A^{\delta^-(v)+1}$ into A built with an s-operator and some r-mappings that modify incoming data. Such a mapping will be used for the local computations by the computing entity v. We generalize binary r-operators to *n-ary r-operators*, which means that incoming data are modified by different r-mappings (Figure 2).

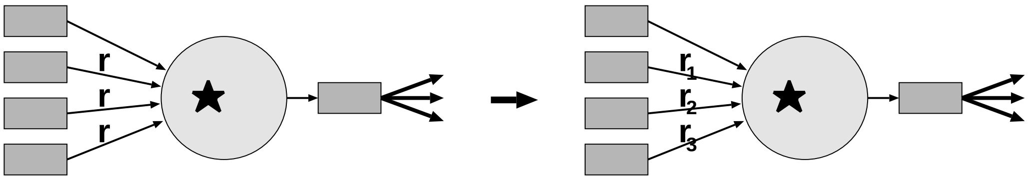

Figure 2: Extension of the binary r-operators to the n-ary r-operators.

Next, we study some more general rs-mappings, and we define *generalized r-operators*, using some more complex r-mappings, and we discuss about hypotheses needed to insure termination of the *direct*−*association*s.

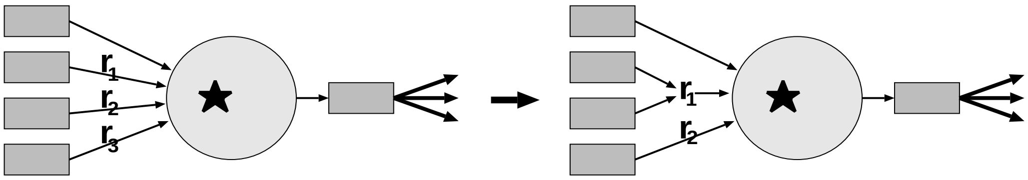

Figure 3: Extension of the n-ary r-operators to the generalized r-operators.

First we give some general graph theory definitions in Section 2. In Section 3, we define the computational model. In Section 4 we recall results about s-operators and r-operators. The n-ary r-operators are presented in Section 5, while generalized r-operators are studied in Section 6. Finally some applications are developed in Section 7.

2 Graph definitions

A *directed graph* or *digraph* $G(V, E)$ is defined with a set of vertices V and a set E of edges (v_1, v_2) which are ordered[1] pairs of vertices of V $(v_1, v_2 \in V)$. The *in-degree* of a vertex v of G, denoted by $\delta^-(v)$ is equal to the number of vertices u such that the edge (u, v) is in E. A *spanning subgraph* $H(V_H, E_H)$ of $G(V, E)$ is a digraph where $V_H = V$ and $E_H \subset E$. A *dipath* from a vertex v_0 to a vertex v_k in $G(V, E)$ is a list of edges of E, $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$. The length of this dipath is k. If each v_i is unique in the dipath, it is said *elementary*. If $v_0 = v_k$, it is called a *cycle*. A digraph without any cycle is called a *directed acyclic graph*. The *distance* between two vertices u, v of G, denoted by $\mathrm{d}_G(u, v)$, is the minimum of the lengths of all dipaths from u to v. The *diameter* of G, denoted by $\mathrm{Diam}(G)$, is the maximum of the distances between all couples of vertices in G. The *strongly connected component* of a vertex v is the set of all vertices $w \in V$ such that there exists a dipath from v to w *and* a dipath from w to v. G is said strongly connected if it has only one strongly connected component.

The *direct descendants* of a vertex v of G are all the vertices w of G such that the edge (v, w) is in E. Their set is denoted by $\Gamma_G(v)$. The *descendants* of v are all the vertices w such that there exists a dipath from v to w. Their set is denoted by $\widehat{\Gamma_G}(v)$. In the same way, the *direct ancestors* of a vertex v of G are all the vertices w of G such that the edge (w, v) is in E. Their set is denoted by $\Gamma_G^{-1}(v)$. The *ancestors* of v are all the vertices w such that there exists a dipath from w to v. Their set is denoted by $\widehat{\Gamma_G^{-1}}(v)$. The *main source* of a digraph G are the vertices s such that there exists a dipath from r to all other vertices.

A *path* from a vertex v_0 to a vertex v_k in $G(V, E)$ is a list of pairs of vertices $(v_0, v_1), \ldots,$ (v_{k-1}, v_k) such that, for each of them, either (v_i, v_{i+1}) is in E, either (v_{i+1}, v_i) is in E. A *tree of* $G(V, E)$ is a digraph $T(V_T, E_T)$ where $V_T \subset V$ and $E_T \subset E$ and such that there is one and only one path between any vertices u and v in T. A *rooted tree* T of G is a tree of G admitting a main source r, called the *root* of the tree. A *rooted spanning tree* $T(V_T, E_T)$ of $G(V, E)$ is a rooted tree of G where $V_T = V$. The root of a rooted spanning tree of G is a main source of G. A *shortest dipath rooted spanning tree* of G is a rooted spanning tree of G such that, for any vertex v, the distance from the root to v in the tree is equal to the distance from the root to v in G $(\mathrm{d}_T(r, v) = \mathrm{d}_G(r, v))$. A *shortest dipath rooted spanning forest* F of G is a set of rooted trees T^i of G, rooted on some roots r^i, such that each vertex v of G belongs to one and only one of these trees. Moreover, if r_j is the root of the tree T_j to which v belongs, $\mathrm{d}_F(r_j, v) = \mathrm{d}_G(r_j, v) \leq \mathrm{d}_F(r_i, v)$ for all roots r_i.

A *weighted digraph* $G_w(V, E)$ is defined as a digraph $G(V, E)$ with an application w from E to $\mathbb{N}$. The *weight* of a dipath P of G_w, denoted by $w(P)$ is the sum of the weights of all its edges. The *weighted distance* between two vertices u and v in G_w, denoted by $\mathrm{d}_{G_w}(u, v)$, is the minimum of the weights of all dipaths from u to v. The *weighted shortest dipath rooted spanning tree* of a weighted digraph G_w is defined similarly to the shortest dipath rooted spanning tree in a digraph G, except that the weighted distance is used instead of the distance. The problem to find such a tree in a weighted digraph is commonly called the *single source shortest path problem*, or *sssp* problem. When there is several sources, we can define similarly the *weighted shortest dipath rooted spanning forest*, and define the *multiple source shortest path problem*, or

[1] $(v_1, v_2) \neq (v_2, v_1)$.

mssp problem.

A *depth-first-search tree* of the digraph G is a rooted spanning tree of G admitting a numbering $f : V \to \mathbb{N}$ of the nodes verifying:

1. f is an injection
2. $\forall (u, v) \in E,\ f(u) < f(v)$
3. $\forall u, w \in V$, if $w \notin \widehat{\Gamma_G}(u)$ then $\forall v \in \widehat{\Gamma_G}(u),\ f(v) < f(w)$

3 Computation model

In this Section, we define associative nets in a more general way than in previous work *(Ducourthial, 1997)*.

3.1 Presentation

The distributed computing model we consider in this work is a set of *computing entities* linked with a *network*. Figure 5 displays such a network with three entities.

A *computing entity* is the smallest unit of hardware or software that performs a calculation (Figure 4). A computing entity v has a limited private memory, $\delta^-(v)+1$ incoming buffers called $b_0(v), b_1(v), \cdots, b_{\delta^-(v)}(v)$ and one outgoing buffer called $b_{\text{out}}(v)$. The order of the incoming buffers is important since it is not always possible to permute incoming data without changing the result.

The *network* is the set of the inter-entities connections (Figure 5). Each link (u, v) of the network is unidirectional, starts from the outgoing buffer $b_{\text{out}}(u)$ of the computing entity u and ends at an incoming buffer $b_i(v)$ $(1 \leq i \leq \delta^-(v))$ of the computing entity v. An incoming buffer is the end of a single link. On each entity, there is a loop, that is an edge from $b_{\text{out}}(v)$ to $b_0(v)$ (Figure 4). Buffers which are at the opposite sides of a link are called *corresponding buffers*.

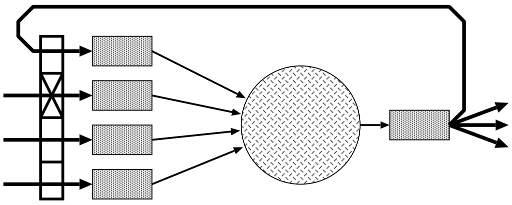

Figure 4: Computing entity.

We suppose that no two entities are linked together with more than one link[2]. Hence the network is modeled by a directed graph without multi-edges (simple digraph, also called digraph in this report).

Communications are asynchronous, which means that the receiver is not waiting for a message, and the sender is not waiting from an acknowledgement of its own message. Moreover the entities are not synchronized. They start a local computation as soon as there is a new data in one of their incoming buffer. As soon as there is a new data in an outgoing buffer, it is sent to all the linked incoming buffers.

[2]Such multi-edges have no interest for the model since forbidding them do not restrict the set of global computations that can be performed. Indeed, suppose that the computing entities u and v are linked with two edges $(u, v)_1$ and $(u, v)_2$. We will see in Sections 4 and 5 that v performs a $\star$ computation (where $\star$ is an s-operator) on incoming data, after they have been modified by r-mappings. Suppose now that datum received by edge $(u, v)_1$ is modified by the r-mapping r_1 and the one received by $(u, v)_2$ by r-mapping r_2. Then these edges can be replaced by a single one (u, v) associated to the mapping $r(x) = r_1(x) \star r_2(x)$. Note that it is still true with more complex r-mappings like those presented in Section 6.

By default, any incoming buffer b_i on any node v contains a special element $e_{i,v}$. Such elements are useful to perform local computations without waiting for incoming data. Local computations on each node v can begin with the initial datum x_v only: in that case, the value $\diamond_v\left(x_v, e_{1,v}, \ldots, e_{\delta^-(v),v}\right)$ is sent on the direct sucessors on the network. That allows to prevent deadlock at the begining of the global computation (where each entity is waiting for incoming data of its direct ancestors).

The elements $e_{i,v}$ can also be used to take into account only a subset of the incoming links. In that case, incoming links are broken in front of their incoming buffers $b_i(v)$ $(0 \leq i \leq n)$. These links and buffers are called *masked*. When a buffer $b_i(v)$ $(1 \leq i \leq n)$ is masked, it contains the element $e_{i,v}$. When the first buffer $b_0(v)$ is masked, it always contains the initial value of the entity.

Figure 5 displays a simple associative net with three computing entities. Note that two buffers are masked.

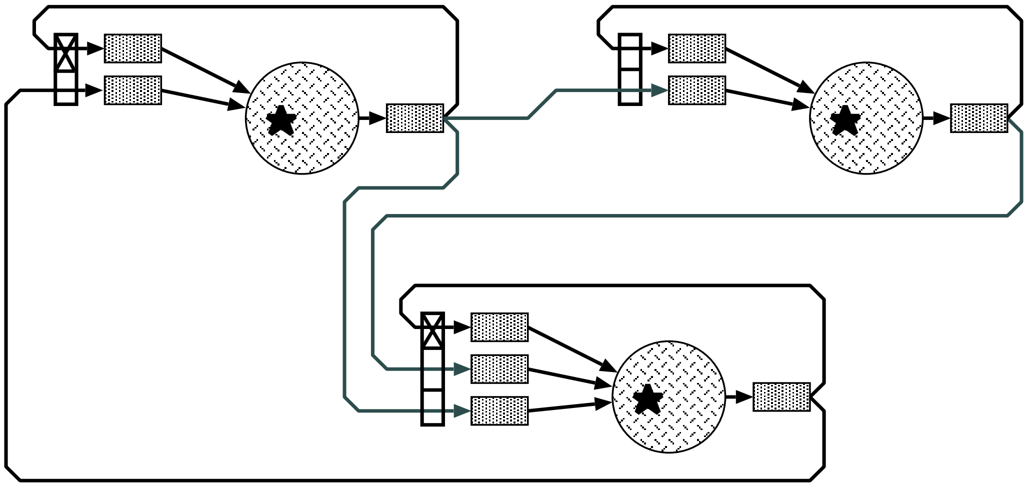

Figure 5: A simple associative net with three computing entities (to buffers are masked).

3.2 Local computation

We call A the set of possible buffer's values, and Δ_v the set of the applications that can be used by the computing entity v to build a datum from those it received in its incoming buffers: Δ_v is the set of all the mappings $\diamond_v$ from $A^{\delta^-(v)+1}$ into A.

We define a *local computation* as the computation performed by a computing entity v. Such a computation is supposed to be atomic: a local computation begins when an incoming buffer is updated with a new value and all new updates of buffers during the computation will be taken into account for the next calculation only. As we saw in previous Section, at the beginning of the global computation, each incoming buffer b_i contains the element $e_{i,v}$. Hence, before first data are received, the local computations use these elements instead of data from direct ancestors. The first buffer b_0 contains the result of the last local computation if it is unmasked, or the initial datum of the entity else. For the first local computation, it contains the initial datum of the entity. We assume that a local computation on v needs δ_v^c units of time. We denote by $b_i^t(v)$ the value of the buffer $b_i(v)$ at the date t. A local computation on v using the mapping $\diamond_v$ of Δ_v is then modeled as follows:

$$b_{\text{out}}^{t+\delta_v^c}(v) = \diamond_v \left(b_0^t(v), b_1^t(v), \ldots, b_{\delta^-(v)}^t(v)\right) \tag{1}$$

Note that all dates used in this report are dates from an external virtual clock, that we use only to study the system. This clock gives the physical time: its value changes even if no event occurs in the network. Such a clock does not exist in the system.

3.3 Global computation

At the beginning of a global computation, initial data are in the incoming buffers b_0 of the network's entities. During the global computation, the network sends data of the outgoing buffers to the corresponding linked buffers, and the computing entities v perform local computations, which means that they build new data from those they receive in their incoming buffers $b_i(v)$, using their mapping $\diamond_v$, and store them into their outgoing buffers $b_{\text{out}}(v)$. When all buffers are stable and no data are in transit, the global computation is ended.

In this report, we admit non-uniform computations: several distinct mappings can be used simultaneously in the network by different computing entities for their local computation during the global computation.

We call this global computation a *direct−association*, although it is an extension of the one defined in *(Ducourthial, 1997)*, which was already slightly different from the *association* defined in *(Mérigot, 1992; Mérigot, 1997)*. If in each computing entity v of the network, the mapping used for local computations is $\diamond_v$, we call $\diamond_G$ the global mapping for the whole network defined using the identifiers[3] of the computing entities as follows:

$$\text{on the computing entity } v, \quad \diamond_G = \diamond_v$$

Such a mapping can be written using a switch-case construction in C language. That allows us to denote, as in previous work, a global computation by $\diamond_G$*−direct−association*. Here, for facilities, we use the identifiers to manipulate the global computation in a SPMD programming style form. This means that the network is used with a Single Program, for Multiple Data (one per node). In this case, we need the switch-case-like construction and then we need the identifiers of the computing entities (non-anonymous computation). However, if the network is used in MPMD programming style (multiple programs for multiple data), the switch-case-like construction is not needed: each computing entity is initialized with its own program, which is different from those of other entities (non-anonymous initialization but anonymous computations).

To conclude, a $\diamond_G$*−direct−association* is a global computation performed by concurrent executions of the local computations based on the $\diamond_v$ mappings, one per computing entity v of

[3]The *identifier*, or the *identity* of a computing entity is its name such that there is a one-to-one mapping between the set of the entities (V) and the set of these identifiers.

V. The result of such a primitive exists only if the global computation terminates. It can be used for programming only if it is deterministic.

3.4 Network properties

3.4.1 General properties

The network updates incoming buffers with values of the outgoing buffers that are linked to. It is necessary, to perform efficient computation, to state that any sent datum is received after a finite time. Without any correction code or protocole, it is also needed to suppose that data are not modified in the links.

Hence, at each times a new datum is copied into $b_{\text{out}}(u)$, then a finite time after, the new value is copied into the corresponding outgoing buffers of the direct successors of u.

For a general purpose network, the hypothesis of unmodified sent messages is not sufficient, and other safe properties on the links are needed *(Raynal and Hélary, 1988; Tel, 1994)*. For example, one can suppose that no sent message is lost, nor duplicated on a link. Moreover, one can suppose that the order of messages successively sent one the same link is not modified (FIFO property).

However, one can remark that if the local computations are performed by an s-operator, then messages can be duplicated or disordered. Moreover, using some simple protocols, one can allow that messages are disordered on a link.

3.4.2 Weak hypothesis for s-computations

Properties of an s-operator $\star$ In Section 4.2, we recall results about s-operators. To prove the previous assertions, one needs to know that an s-operator is associative, commutative and idempotent. Moreover it defines a partial order relation.

Since $\star$ is idempotent, the global computation is unchanged if all buffers b_0 are unmasked. In that case, the successive values built by each computing entity are more and more small. Indeed, an s-computation[4] is always smaller than any of its variables, and the last result of the entity is one of the variables of the s-computation on the entity when the buffers b_0 are unmasked. In the following, we suppose that buffers b_0 are unmasked.

If $\star$ defines a total ordering on A, such as the **min** operator on $\mathbb{N}$, then the result of a $\star$*−direct−association* on each computing entity is the smallest received value. If the ordering relation is not total, then the result is the composition by $\star$ of all the received data, which is smaller than all the received data. However, in both cases, the successive values built by each entities are more and more small, in sense of the partial ordering relation induced by $\star$, as exlpained above.

[4] An s-computation is a computation with an s-operator.

Dupplication Consider an associative nets where the local computations are based on the s-operator $\star$, and let (v, w) be a link in the network. If $b_{\text{out}}(v)$ contains the value x, the expression built by v must contain x. Since the communication delay is finite, v receives x a finite time after the begining of the global computation.

Suppose now that w receives several times the value x. Then, using associativity of $\star$, the expression in $b_{\text{out}}(w)$ can be written without any bracket. Next, using commutativity of $\star$, these values can be placed on the head of the expression of $b_{\text{out}}(w)$. Finally, using idempotency of $\star$, the doubles can be suppressed, and the expression is equivalent to those obtained if x is received only one times by w.

Disordering In the same network, suppose now that $b_{\text{out}}(v)$ first contains x and some times latter it contains x'. Following remarks of Paragraph Properties about decreasing of the successive results, we have $x' \preceq_\star x$, where $\preceq_\star$ is the partial order relation induced by $\star$, defined by: $x' \preceq_\star x \overset{\text{def}}{\Leftrightarrow} x' \star x = x'$.

Suppose that x is received first by w. When x' is coming on w, a new expression is built by the entity and stored in $b_{\text{out}}(w)$. Due to the loop (b_0 is unmasked), this expression both x and x' (Figure 6). So it is equivalent to the expression without x. Indeed, if x and x' are both present in the expression, then first the expression can be written without bracket, second x and x' can be isolated at the head of the expression, and third x can be suppressed because $x' \star x = x'$.

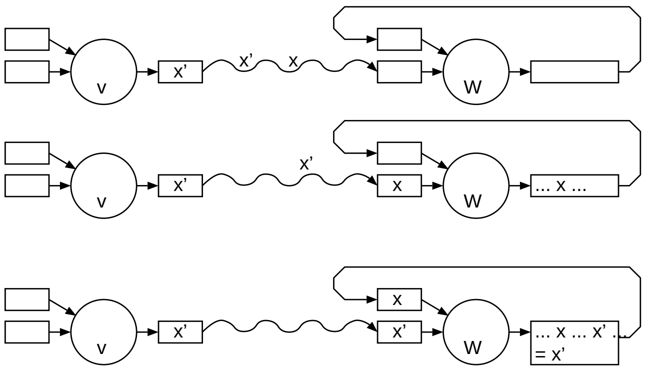

Figure 6: As soon as x' comes to w, the expression in $b_{\text{out}}(w)$ can be written without x.

Suppose now that x' is received first by w and some times latter x is coming (Figure 7). The new expression built by w contains both x and x', then the result is similar to the one without x because, as in previous case, x can be suppressed.

Loss Since dupplication of messages are allowed, one can imagine a simple protocole to implement the safe hypothesis. It consists in systematically resending data after a delay. This protocol can be used in associative nets where the network is not a shared resource: a link between the computing entity v and the computing entity w is used only to send data from v to w. In that case, these links can be used as many as possible to update buffers, since unecessary communication won't slow the global computation with network contention. Such networks

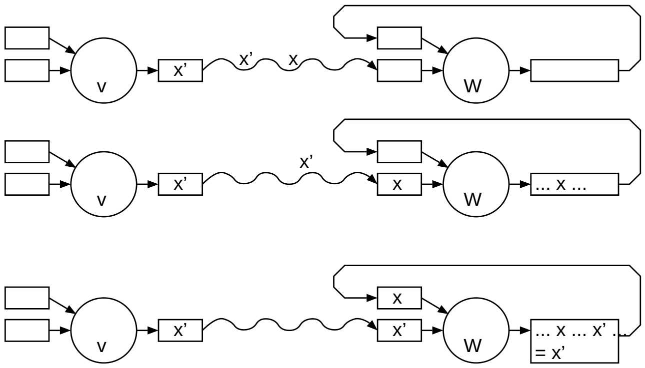

Figure 7: If x' comes first to w, the result is unchanged when x comes to w.

exist in special parallel computers with dedicated interconnexion networks, like the Associative Mesh *(Dulac, 1996)*.

3.4.3 Weak hypotheses for r-operators

A local computation with an r-operator is based on an s-computation. The only difference with s-operators is due to the modifications of the incoming data with r-mappings, just before the s-computation. However such modifications by r-mappings do not unvalidate the previous discussion about hypotheses of the links in the network. Hence, when local computations use binary, n-ary or generalized r-operators (the difference is in the kind of the r-mappings), data can be dupplicated and disordered. Moreover, they can be lost if the network is not a shared resource.

3.5 Termination of a direct−association

A *direct−association* needs a termination detection system to be used in a program. In this report, our work does not deal with such a system. In *(Ducourthial, pear)*, we proposed a slightly modification of *direct−association*s that allow to detect their termination. In practice, to obtain efficient distributed computations, some hypotheses on the time spent by data on the links have been made (we admitted Asynchronous Bounded Delay hypotheses *(Tel, 1994)*).

It is convenient to consider the $\diamond_G$−*direct−association* primitive as a mechanism to build regular expressions with brackets, $\diamond_v$ mappings ($v \in V$) and data of A which were in the network at the beginning of the global computation. The result of any global computation is a set of such expressions, one per computing entity, which may be of infinite length. We denote expressions with the letter X, and we note them between square brackets $X = [\ldots]$. We use a wild card: the notation $[exp]^+$ holds for a non-null number of times the expression $[exp]$: $X = [a]^+ = [aa\ldots a]$. Moreover, we assume that all expressions are well bracketed, even if some wild cards containing brackets are used.

As an example, in Figure 8 where a $\diamond_G$−*direct−association* is performed, we call $X_f(u)$ the final expression (result) of the computing entity u and $X_f(v)$ the one of the entity v. f is the first date such that for any date $f' > f$, $X_{f'}(u) = X_f(u)$ and $X_{f'}(v) = X_f(v)$. At the date f, the $\Diamond_G-\textsf{direct}-\textsf{association}$ gives the following expressions:

$$X_f(u) \quad = \quad [\Diamond_u(x)]$$

The successive expressions for the second entity are:

$$\begin{array}{l} [\Diamond_v(y, e_{1,v}, e_{2,v})], \\ [\Diamond_v(y, e_{1,v}, \Diamond_v(y, e_{1,v}, e_{2,v}))], \\ [\Diamond_v(y, e_{1,v}, \Diamond_v(y, e_{1,v}, \Diamond_v(y, e_{1,v}, e_{2,v})))], \\ \ldots, \\ [[\Diamond_v(y, e_{1,v},]^+ e_{2,v}[)]+] \end{array}$$

as long as no datum is received from the entity u. If the expression in the entity v is X when the datum $\Diamond_u(x)$ is received from the entity u, the successive expressions on v are:

$$\begin{array}{l} X, \\ [\Diamond_v(y, \Diamond_u(x), X)], \\ [\Diamond_v(y, \Diamond_u(x), \Diamond_v(y, \Diamond_u(x), X))], \\ [\Diamond_v(y, \Diamond_u(x), \Diamond_v(y, \Diamond_u(x), \Diamond_v(y, \Diamond_u(x), \Diamond_v(y, \Diamond_u(x), X))))], \\ \ldots, \\ [[\Diamond_v(y, \Diamond_u(x),]^+ X[)]^+] \end{array}$$

Hence the regular expression $X_f(v)$ that characterize the result on the entity v is

$$X_f(v) \quad = \quad [[\Diamond_v(y, \Diamond_u(x),]^+ [\Diamond_v(y, e_{1,v},]^+ e_{2,v}[)]+]$$

If this global computation terminates, there is a deterministic way to simplify — or reduce — the expression $X_f(v)$, using the properties of the mappings $\Diamond_u$ and $\Diamond_v$ such that $X_f(v)$ is equal to a finit length deterministic expression.

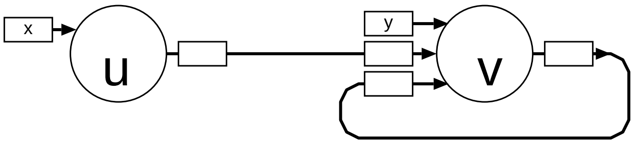

Figure 8: An example of expressions built with the network: $X_f(u) = [\Diamond_u(x)]$ and $X_f(v) = [[\Diamond_v(y, \Diamond_u(x),]^+ [\Diamond_v(y, e_{1,v},]^+ e_{2,v}[)]+]$.

More formally, we define the termination of a $\textsf{direct}-\textsf{association}$ as follows.

Definition 1 *The distributed computation $\Diamond_G-\textsf{direct}-\textsf{association}$ based on the mappings $\Diamond_v \in \Delta_v$ for each computing entity v* terminates on a deterministic result on any network and with any data *if, for any network $G(V, E)$ and any data of A, for any execution performed on this network and with these data, there exists a date t such that for any computing entity v of the network, we have $X_t(v) = X_{t+\tau}(v) = X(v)$, for any $\tau > 0$, where $X_t(v)$ is the expression on v at the date t. In that case, $X(v)$ is called the* result on v of the $\Diamond_G-\textsf{direct}-\textsf{association}$.

When the $\diamond_G$ – *direct – association* terminates on a deterministic result with any data of A, this global computation defines a mapping from $A^{|V|}$ into $A^{|V|}$. Indeed, in that case, the $\diamond_G$–*direct–association* transforms the initial data of the $|V|$ computing entities of the network into $|V|$ final data on these entities.

In this report, we study properties of operators that insure the termination of the *direct–associations* on any network and with any data of their definition set A.

3.6 Complexity of a direct – association

In this Section, we suppose that the global computations are deterministic, which means that if the global computation is performed several times on the same network initialized with the same data, then the same result is always obtained. This assumption is not so strong since undeterministic global computations are not interesting for programming in general.

3.6.1 Time complexity

For the purpose of the time complexity measure, we assume the *Asynchronous Bounded Delay (ABD)* hypotheses *(Tel, 1994)*, which means that any communication delay is bounded by δ_s. Hence, any sent datum is received before δ_s units of times (from an external virtual clock). Without such hypothesis, it is impossible to study the time complexity.

More, we state that any local computation needs at most δ_c units of times. However, some more accurate bounds on the time complexity could be obtained one local bound is used per entity.

The time complexity of a *direct–association*, *i.e.* the time needed to obtain the termination of a $\diamond_G$ – *direct – association*, can be decomposed into two steps:

<u>building</u> first the final result of any entity must be built somewhere in the network,
<u>diffusing</u> second, the final result of any entity must be received by the entity itself.

Diffusing step. Since we assume that a sent data is received after δ_s units of times, and that any local computation needs at most δ_c units of times, the time complexity of the diffusing step is at most $\text{Diam}(G) \times (\delta_s + \delta_c)$ (traversing one edge needs a computation and a sending): as soon as the last final result has been built, it can reach all the network after $\text{Diam}(G)$ communication steps, that needs at most $\delta_s + \delta_c$ units of time each, because there is no contention betwwen final data.

Indeed, suppose that the final datum $d(A)$ of the node A has been built on the node a, the final datum $d(B)$ of the node B has been built on the node b and the paths from a to A and from b to B have a common segment in the network (see Figure 9). Since we supposed that the global computation is deterministic, the same global result is obtained when $d(A)$ first cross the common segment or when $d(B)$ first cross the common segment. Since $d(A)$ is the final datum of A, it will not be modified during its transit from a to A. This means that any node in the common segment has the datum $d(A)$ in their outgoing buffers after $d(A)$ has crossed the common segment. Similarly, when $d(B)$ cross the common segment, it is not modified and the nodes of the segment have the datum $d(B)$ in their outgoing buffers. Now consider the first node v of the common segment and suppose that $d(A)$ and $d(B)$ come together to v. The result of the local computation on v is equal to $d(A)$ and to $d(B)$. Hence $d(A) = d(B)$, and the final data $d(A)$ and $d(B)$ are not slowed during the diffusing step because they are identical. The diffusing step is then bounded by $\mathrm{Diam}(G) \times (\delta_s + \delta_c)$.

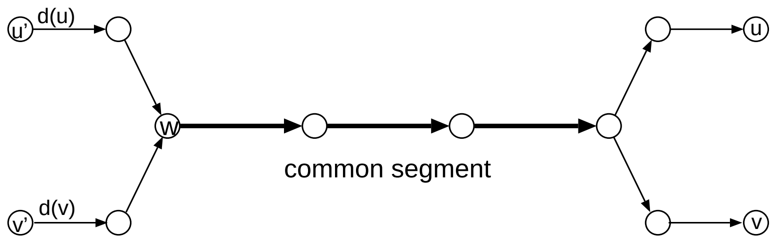

Figure 9: If $d(A)$ and $d(B)$ are final results of A and B respectively, then $d(A) = d(B)$ and the diffusing step is bounded by $\mathrm{Diam}(G) \times (\delta_s + \delta_c)$.

Building step. The time needed by the building step can not be evaluate in the general case for all the operators. Indeed, it depends intrinsically of the operator and of the initial data. By instance, consider the operator $(x, y) \mapsto \mathbf{min}(100, x + 1, y + 1)$ on the associative net of Figure 10, initialized with zero in the three entities. The final result on each entity is 100. Here, the time complexity of the building step is equal to $99\delta_s + 100\delta_c$, where δ_c is the time needed to perform one local computation and δ_s the time needed to send one data (supposing these times are constant in the network).

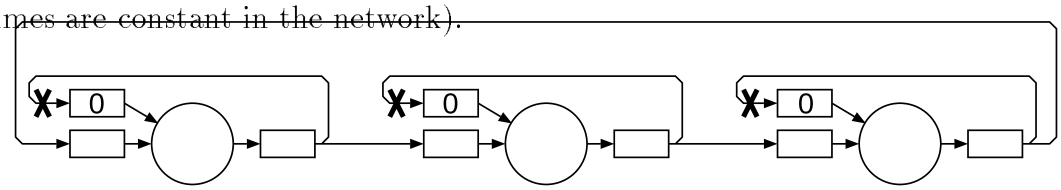

Figure 10: The time complexity of the building step depends on the operator and on the initial data: consider the operator $(x, y) \mapsto \mathbf{min}(100, x + 1, y + 1)$.

Time complexity for s-operators The result of a $\star$*—direct—association* is given in Theorem 4 page 18. For any computing entity, it is equal to the s-computation on the values of the ancestors of the entity. This result is built on the entity itself. Hence the diffusing step is null. The result is obtained as soon as all the data of the ancestors are received. There is no contention in the network between the values that must receive the entities. Indeed, if two data x and y must take the same link, then the datum $x \star y$ will be sent. Hence all the data are received after $\mathrm{Diam}(G) \times (\delta_s + \delta_c)$. Hence, the time complexity of a $\star$*— direct — association* is $\mathrm{Diam}(G) \times (\delta_s + \delta_c)$.

Note that when the s-operator defines a total ordering on A, then the final result of each entity is the smallest value of all those of its ancestors. In that case, the building step is null and the diffusing step needs at most $\mathrm{Diam}(G) \times (\delta_s + \delta_c)$, as explained previously.

However the termination can hold more quickly in many cases. For example, with the uniform global computation **and** − *direct* − *association* on boolean data, the time needed for termination is equal to $d \times (\delta_c + \delta_s)$, where d is the maximum distance between a computing entity with initial datum 1 and another one with initial datum 0 (supposing they both exist). Simulations have showed that a uniform *direct*−*association* takes less than $\mathrm{Diam}(G) \times (\delta_c + \delta_s)$ time units in most practical cases (for example in image analysis *(Dulac, 1996)*).

Time complexity of r-operators For an r-operator (see Section 4.2), the diffusing step is immediate since the final result of any computing entity is built on the entity itself. The time complexity of a *direct* − *association* is then equal to the time complexity of the building step, which needs, as for s-operators, at most $\mathrm{Diam}(G) \times (\delta_c + \delta_s)$ units of times (see Theorem 10 page 19).

As for s-operators, the average time complexity could be very smaller than this upper bound, depending on the initial data.

3.7 Message complexity

Message complexity is a classical measure of performance in distributed system. It consists in computing the maximal number of messages needed to achieve the global computation. However, it does not reflect the topology of the network, which can induce some conflicts on the link, that slows the global computation. Moreover, it can be useful to send more messages than necessary to increase time complexity. See discussion in Section 3.4.2 about networks that are not shared resources (*i.e.* there exists one physical link for any edge of the network).

We always privilege the time complexity instead of the message complexity.

The message complexity can not be evaluated for any operator (see "Building step" page 16). One can remark that after each local computation on v, there are $\delta^+(v)$ messages sent, where $\delta^+(v)$ denote the out-degree of the node v. When the time complexity is bounded by $\mathrm{Diam}(G) \times (\delta_c + \delta_s)$, the message complexity can be bounded by $|E| \times \mathrm{Diam}(G) \times (\delta_c + \delta_s)$. However the *direct* − *association*s are performed on spanning subgraphs (with less edges than the whole network) in most practical cases.

4 Previous results

In this section we present two classes of operators. More informations can be found in *(Ducourthial, 1997)*. In the following Δ is the set of binary operators on the set of data A respecting the following axioms:

(1) The operators of Δ are internal laws in A: $\forall\diamond \in \Delta, \forall x, y \in A, \quad \exists! z \in A$ such that $x \diamond y = z$.

(2) The operators of Δ are "left-associative" on A and we use the following convention to simplify the writing of expressions: $\forall\diamond \in \Delta \quad (x \diamond y) \diamond z = x \diamond y \diamond z$.

(3) Each operator of Δ admits a right-identity element: $\forall\diamond \in \Delta \quad \exists e_\diamond \in A$, such that $\forall x \in A, \quad x \diamond e_\diamond = x$.

(4) Each operator $\diamond$ of Δ allows left-cancellation: $\forall\diamond \in \Delta, \forall y, z \in A, \quad (\forall x \in A, \quad x \diamond y = x \diamond z) \quad \Leftrightarrow \quad (y = z)$.

4.1 s-operators

Definition 2 *A binary operator $\star$ is an* s-operator *on A if it is associative, commutative and idempotent on A: for all x, y and z in A, $(x \star y) \star z = x \star (y \star z)$, $x \star y = y \star x$, and $x \star x = x$.*

It can be shown that an s-operator defines a partial ordering. This result is known as the Infimum Theorem *(Tel, 1991)*. That is why we call such an operator an s-operator: s holds for *sort*. We denote by $\preceq_\star$ the partial ordering relation induced by $\star$ and by $\perp_\star\{x, y\}$ the largest element of the subset of A containing all elements z such that $z \preceq_\star x$ and $z \preceq_\star y$ (supposing it exists, see theorem below).

Theorem 3 (Infimum Theorem) *Let $\star$ be an s-operator on A. Then the binary relation $\preceq_\star$ defined by $(x \preceq_\star y) \equiv (x \star y = x)$ is a partial ordering on A. Moreover $\perp_\star\{x, y\} = x \star y$.*

When it exists, the identity element e of the s-operator $\star$ is the unique element of A which verifies $x \preceq_\star e$ for any x in A. If there is no such larger element in A, one can always build it and add it to A. As explained in Section 3, this element allows to mask some incoming links and to begin local computations before all incoming data are received. One can easily see that an s-operator verifies the axioms of Δ.

The following theorem is proved in *(Ducourthial, 1997)*. The notation $\star\{x_0, \ldots, x_n\}$ holds for $x_0 \star x_1 \star \ldots \star x_n$ (well defined when $\star$ is associative and commutative, as for instance for the s-operators).

Theorem 4 *If $\star$ is an s-operator on A, then the global computation $\star-\mathsf{direct}-\mathsf{association}$ terminates on a deterministic result in any network G using any data of A, so that the result on each computing entity v is $\star\left\{b_0^0(u), u \in \widehat{\Gamma_G^{-1}}(v)\right\}$*

4.2 r-operators

Definition 5 *A binary operator $\triangleleft$ is an* r-operator *on A if there exists a bijective mapping r from A to A, called an* r-mapping, *such that for all x, y and z in A:*

$$\begin{array}{lcll} (x \triangleleft y) \triangleleft r(z) & = & x \triangleleft (y \triangleleft z) & (r\textit{-associativity}) \\ r(x) \triangleleft y & = & r(y) \triangleleft x & (r\textit{-commutativity}) \\ r(x) \triangleleft x & = & r(x) & (r\textit{-idempotency}) \end{array}$$

Notice that we always call such an operator an r-operator, even if the mapping is not called r (r holds for *reduce* in the term r-operator). In the following, we considere r-operators of Δ. Here are some properties of r-operators proved in *(Ducourthial, 1997)*.

Proposition 6 *The bijective mapping r is an isomorphism of $(A, \triangleleft)$.*

Theorem 7 *If $\triangleleft$ is an r-operator on A with a bijective mapping r from A to A, then the operator $\star$ defined by $x \star y = x \triangleleft r^{-1}(y)$ is an s-operator on A.*

The identity element of $\star$ is the right-identity element of $\triangleleft$. This element also allows to mask some incoming links and to begin local computations before all incoming data have been received (see Section 3).

Proposition 8 *The mapping r is an isomorphism of $(A, \star)$.*

Theorem 9 *If for all x and y in A, the binary operators $\triangleleft$ and $\star$ verify $x \triangleleft y = x \star r(y)$ with r a mapping from A to A then the three following propositions are equivalent:*
(1) r is a surjective mapping from A to A, the operator $\triangleleft$ has a right-identity element, it allows left-cancellation and it is r-commutative, r-associative and r-idempotent on A,
(2) $\triangleleft$ is an r-operator defined on A which allows left-cancellation and admits a right-identity element,
(3) $\star$ is an s-operator on A admitting an identity element and $r : A \to A$ is an isomorphism of $(A, \star)$.

The r-operator $\triangleleft$ is said idempotent if $x \triangleleft x = x$, which is equivalent to $x \preceq_\star r(x)$.

Theorem 10 *If $\triangleleft$ is an idempotent r-operator on A, such that for any x, y in A, $x \triangleleft y = x \star r(y)$, with $\star$ an s-operator on A and r an isomorphism of $(A, \star)$, then the global computation* $\triangleleft-$**direct**$-$**association** *terminates on a deterministic result in any network G using any data of A, so that the result on each node v is $\star \left\{ r^{\mathrm{d}_G(u,v)}(b_0^0(u)), u \in \widehat{\Gamma_G^{-1}}(v) \right\}$*

5 n-ary r-operators

5.1 Definitions

An r-operator $\triangleleft$ can simply be defined on A from an s-operator $\star$ and an isomorphism r of $(A, \star)$ as follows: for all x, y in A, $x \triangleleft y = x \star r(y)$ (Theorem 7). As an example, the binary r-operator **minc** has been defined on $\overline{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, +\infty\}$ by $\mathbf{minc}(x, y) = \mathbf{min}(x, y + 1)$, where the s-operator is **min** and the r-operator is $x \mapsto r(x) = x + 1$.

Following this construction of binary r-operators, we define the $\mathbf{minc}_w$ n-ary operator on $\overline{\mathbb{Z}}$:

$$\mathbf{minc}_w\,(x_0, \ldots, x_{n-1}) = \mathbf{min}\,(x_0, x_0 + w_1, \ldots, x_{n-1} + w_{n-1})$$

More generally, we define n-ary r-operators as follows.

Definition 11 *A mapping $\bigtriangledown$ from A^n into A is called a* n-ary r-operator *if there exists an s operator $\star$ on A and $n-1$ isomorphisms r_i of $(A, \star)$ such that for all x in A,*

$$\bigtriangledown(x_0, \ldots, x_{n-1}) = x_0 \star r_1(x_1) \star \ldots \star r_{n-1}(x_{n-1})$$

In other words, an n-ary r-operator is built with $n-1$ r-operators $\triangleleft_i$ based on the same s-operator and the r-mappings r_i as follows: $x \triangleleft_i y = x \star r_i(y)$. If all these $n-1$ r-operators $\triangleleft_i$ are idempotent ($x \triangleleft_i x = x \star r_i(x) = x$), we say that the n-ary r-operator is idempotent.

Definition 12 *If $\bigtriangledown$ is an n-ary r-operator on A built with the s-operator $\star$ and the $n-1$ isomorphisms r_i so that for all i $(1 \leq i < n)$, for all x in A, $x \star r_i(x) = x$, then $\bigtriangledown$ is called* idempotent n-ary r-operator.

For the operator $\mathbf{minc}_w$, the $n-1$ r-mappings are $x \mapsto x + w_i$ $(1 \leq i \leq n-1)$. They are isomorphisms of $(\overline{\mathbb{Z}}, \mathbf{min})$. If each w_i is larger than 0, we have $\mathbf{min}(x, x + w_i) = x$ and then $\mathbf{minc}_w$ is said idempotent.

Let (u, v) be an edge of an associative net where the local computations on each node v are based on a $(\delta^-(v)+1)$-ary r-operator built with the s-operator $\star$. Then for each edge (u, v) an r-mapping is given, that we denote[5] by r_v^u (Figure 11).

Similarly, a mapping can be defined for any dipath in G (Figure 12).

Definition 13 *Let $P = (v_0, v_1) \ldots (v_{k-1}, v_k)$ be a dipath of the digraph that models an associative net where the local computations on each computing entity v are based on an $(\delta^-(v)+1)$-ary r-operator built with the s-operator $\star$. The mapping r_P from A into A defined by*[6]

$$r_P(x) = r_{v_k}^{v_{k-1}} \circ \cdots \circ r_{v_0}^{v_1}(x) = r_{v_k}^{v_{k-1}}(\cdots(r_{v_0}^{v_1}(x))\cdots)$$

is called the r-dipath-mapping *of P.*

[5] This is possible because there is no more one edge between any pair of vertices (see Section 3).

[6] French notation of the composition of applications: $f \circ g(x) = f\,(g(x))$.

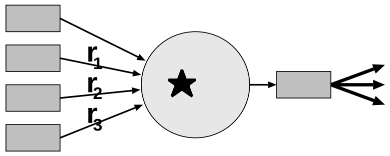

Figure 11: There is one r-mapping per edge.

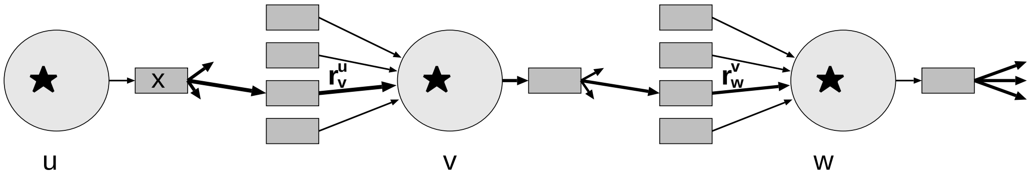

Figure 12: w includes the value $r^v_w \circ r^u_v(x)$ in its local compuation; $r^v_w \circ r^u_v$ is an r-dipath-mapping.

5.2 General properties

Since an n-ary r-operator is based on $n-1$ r-operators built with the same s-operator $\star$ and some r-mappings r_i, some of the properties of the binary r-operators can be extended to n-ary r-operators. First, we extend properties of binary operators of Δ into properties of n-ary r-operators[7] (see axioms of Δ in Section 4).

Proposition 14 *If $\bigtriangledown$ is an n-ary r-operator built with the s-operator $\star$ and the $n-1$ r-mappings r_i, then it verifies the following properties:*

n-ary internal law: $\forall x_i \in A,\ i \in \{0, \dots, n-1\}$, $\exists! z \in A$ *such that* $\bigtriangledown(x_0, \dots, x_{n-1}) = z$

n-ary right-identity element: $\forall x \in A$, $\bigtriangledown(x, e, \dots, e) = x$, *where e is the identity element of $\star$.*

n-ary left-associativity: $\forall x_i \in A,\ \forall k,\ 0 \le i, k \le n-1$,
$$\bigtriangledown\left(\bigtriangledown(x_0, \dots, x_k, e, \dots, e), e, \dots, e, x_{k+1}, \dots, x_{n-1}\right) = \bigtriangledown(x_0, \dots, x_{n-1})$$

n-ary left-cancellation: $\forall y, z \in A,\ \forall k,\ 1 \le k \le n-1$,
$$\left(\begin{array}{l} \forall (x_0, \dots, x_{k-1}, x_{k+1}, \dots, x_{n-1}) \in A^n \\ \bigtriangledown(x_0, \dots, x_{k-1}, y, x_{k+1}, \dots, x_{n-1}) = \\ \bigtriangledown(x_0, \dots, x_{k-1}, z, x_{k+1}, \dots, x_{n-1}) \end{array}\right) \Longrightarrow (y = z)$$

Proof. The three first properties are immediate. For the forth, it is due to the fact that $\star$ allows the left-cancellation and that each r_i is a bijective mapping (isomorphism of $(A, \star)$). □

One can easily verify that these properties holds for $n = 2$, *i.e.* for a binary r-operator $\triangleleft$ such as **minc**: $\mathbf{minc}(x, y) = x \triangleleft y = z \in A$, $\mathbf{minc}(x, e) = x \triangleleft e = e$ and $\mathbf{minc}(\mathbf{minc}(x, e), y) = (x \triangleleft e) \triangleleft y = \mathbf{minc}(x, y)$. Next, there are properties[8] equivalent to those used to define r-

[7] To simplify the comparisons, we conserve the same names with the prefix "n-ary".

[8] As for previous proposition, we conserve the binary names to simplify comparisons, and we add the prefix "n-ary".

operators. The following proposition is immediate from the definition of the n-ary r-operators, using the properties of $\star$ (associative, commutative and idempotent).

Proposition 15 *If $\bigtriangledown$ is an n-ary r-operator built with the s-operator $\star$ and the $n-1$ r-mappings r_i, then it verifies the following properties:*

n-ary r-associativity: $\forall x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1} \in A$, $\forall i \in \{1, \ldots, n-1\}$,
$$\bigtriangledown(x_0, \ldots, x_{i-1}, {r_i}^{-1}(\bigtriangledown(y_0, \ldots, y_{n-1})), x_{i+1}, \ldots, x_{n-1})$$
$$= \bigtriangledown(x_0 \star y_0, \ldots, x_{i-1} \star y_{i-1}, y_i, x_{i+1} \star y_{i+1}, \ldots, x_{n-1} \star y_{n-1})$$

n-ary r-commutativity: $\forall x_0, \ldots, x_{n-1} \in A$, $\forall i, j \in \{1, \ldots, n-1\}$,
$$\bigtriangledown(x_0, \ldots, x_{i-1}, {r_i}^{-1}(x_i), x_{i+1}, \ldots, x_{j-1}, {r_j}^{-1}(x_j), x_{j+1}, \ldots, x_{n-1})$$
$$= \bigtriangledown(x_0, \ldots, x_{i-1}, {r_i}^{-1}(x_j), x_{i+1}, \ldots, x_{j-1}, {r_j}^{-1}(x_i), x_{j+1}, \ldots, x_{n-1})$$

and for $i = 0$:
$$\bigtriangledown(x_i, \ldots, x_{j-1}, {r_j}^{-1}(x_j), x_{j+1}, \ldots, x_{n-1})$$
$$= \bigtriangledown(x_j, \ldots, x_{j-1}, {r_j}^{-1}(x_i), x_{j+1}, \ldots, x_{n-1})$$

n-ary r-idempotency: $\forall x_0, \ldots, x_{n-1} \in A$, $\forall i \in \{1, \ldots, n-1\}$,
$$\bigtriangledown(x_0, \ldots, x_{i-1}, {r_i}^{-1}(r_j(x_j)), x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_{n-1})$$
$$= \bigtriangledown(x_0, \ldots, x_{i-1}, e, x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_{n-1})$$
$$= \bigtriangledown(x_0, \ldots, x_{i-1}, {r_i}^{-1}(r_j(x_j)), x_{i+1}, \ldots, x_{j-1}, e, x_{j+1}, \ldots, x_{n-1})$$

and for $i = 0$:
$$\bigtriangledown(r_j(x_j), \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_{n-1})$$
$$= \bigtriangledown(e, x_1, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_{n-1})$$
$$= \bigtriangledown(r_j(x_j), x_1, \ldots, x_{j-1}, e, x_{j+1}, \ldots, x_{n-1})$$

As for Proposition 14, these n-ary properties holds when $n = 2$ for a binary r-operator $\lhd$, such as **minc**: $\mathbf{minc}(x, r^{-1}(\mathbf{minc}(y, z))) = x \star (y \star r(z)) = x \star y \star r(z) = \mathbf{minc}(x \star y, z)$, $\mathbf{minc}(x, r^{-1}(y)) = x \star y = y \star x = \mathbf{minc}(y, r^{-1}(x))$ and $\mathbf{minc}(r(x), x) = r(x) \lhd x = r(x)$.

For the reciprocity (*i.e.* to obtain an n-ary r-operator from these properties), we need another property, that we call r-isomorphism property (see Lemma 17 below). We first give two lemmas. Reciprocity is given by Theorem 18.

Lemma 16 *If $\bigtriangledown$ is a mapping from A^n to A such that there exists a binary operator $\star$ and $n-1$ bijective mappings r_i verifying, for all $(x_0, \ldots, x_{n-1})$ in A^n, $\bigtriangledown(x_0, \ldots, x_{n-1}) = x_0 \star r_1(x_1) \star \ldots \star r_{n-1}(x_{n-1})$, and if $\bigtriangledown$ verifies the properties of n-ary r-associativity, n-ary r-commutativity and n-ary r-idempotency on A, then $\star$ is an s-operator on A.*

Proof. For all x, y and z in A, we have
$x \star (y \star z) = \bigtriangledown(x, {r_1}^{-1}(y \star z), e, \ldots, e) = \bigtriangledown(x, {r_1}^{-1}(\bigtriangledown(y, {r_1}^{-1}(z), e, \ldots, e)), e, \ldots, e)$, which is equal to (using n-ary r-associativity) $\bigtriangledown(x \star y, {r_1}^{-1}(z), e, \ldots, e) = (x \star y) \star z$. Thus $\star$ is associative on A.

For all x and y in A, we have $x \star y = \bigtriangledown(x, {r_1}^{-1}(y), e, \ldots, e) = \bigtriangledown(y, {r_1}^{-1}(x), e, \ldots, e)$ using n-ary r-commutativity. Thus $x \star y = y \star x$ and $\star$ is commutative on A.

For all x in A, we have $r_1(x) \star r_1(x) = \bigtriangledown(r_1(x), x, e, \ldots, e) = \bigtriangledown(r_1(x), e, \ldots, e)$ using n-ary r-idempotency. Thus $r_1(x) \star r_1(x) = r_1(x)$ and $\star$ is idempotent on A because r_1 is a bijective mapping. $\square$

Lemma 17 *If $\bigtriangledown$ is a mapping from A^n to A such that there exists a binary operator $\star$ and $n-1$ bijective mappings r_i verifying, for all $(x_0, \ldots, x_{n-1})$ in A^n, $\bigtriangledown(x_0, \ldots, x_{n-1}) = x_0 \star r_1(x_1) \star \ldots \star r_{n-1}(x_{n-1})$, and if each mapping r_i verify the r-isomorphism property defined by*

$$r_i\left(\bigtriangledown\left(x_0, {r_1}^{-1}(x_1), \ldots, {r_{n-1}}^{-1}(x_{n-1})\right)\right) = \bigtriangledown\left(r_i(x_0), {r_1}^{-1} \circ r_i(x_1), \ldots, {r_{n-1}}^{-1} \circ r_i(x_{n-1})\right)$$

then each mapping r_i is an isomorphism of $(A, \star)$.

Proof. For all $(x_0, \ldots, x_{n-1})$ in A^n, we have $r_i\left(\bigtriangledown\left(x_0, {r_1}^{-1}(x_1), \ldots, {r_{n-1}}^{-1}(x_{n-1})\right)\right) = r_i(x_0 \star \ldots \star x_{n-1})$. Moreover $\bigtriangledown\left(r_i(x_0), {r_1}^{-1} \circ r_i(x_1), \ldots, {r_{n-1}}^{-1} \circ r_i(x_{n-1})\right) = r_i(x_0) \star \ldots \star r_i(x_{n-1})$. Thus $r_i(x_0 \star \ldots \star x_{n-1}) = r_i(x_0) \star \ldots \star r_i(x_{n-1})$ and r_i is an isomorphism of $(A, \star)$. $\Box$

Using the two preceding lemmas, we can now show the following theorem.

Theorem 18 *Let $\bigtriangledown$ be a mapping from A^n to A such that there exists $n-1$ bijective mappings r_i and a binary operator $\star$ verifying, for all $(x_0, \ldots, x_{n-1})$ in A^n, $\bigtriangledown(x_0, \ldots, x_n) = x_0 \star r_1(x_1) \star \ldots \star r_{n-1}(x_{n-1})$. Then the two following propositions are equivalent:*
(1) $\bigtriangledown$ is a n-ary r-operator, i.e. $\star$ is an s-operator on A and each mapping r_i is an isomorphism of $(A, \star)$.
(2) $\bigtriangledown$ verifies the properties of n-ary r-associativity, n-ary r-commutativity, n-ary r-idempotency (defined in Proposition 15) and r-isomorphism (defined in Lemma 17).

Proof. (2) $\Rightarrow$ (1) is given by Lemmas 16 and 17.

(1) $\Rightarrow$ (2) is given by Proposition 15 and by the fact that if each r_i is an isomorphism of $(A, \star)$ then $\bigtriangledown$ verifies the r-isomorphism property: $r_i(x_0 \star \ldots \star x_{n-1}) = r_i(x_0) \star \ldots \star r_i(x_{n-1})$ implies that $r_i(\bigtriangledown(x_0, {r_1}^{-1}(x_1), \ldots, {r_{n-1}}^{-1}(x_{n-1}))) = \bigtriangledown(r_i(x_0), {r_1}^{-1} \circ r_i(x_1), \ldots, {r_{n-1}}^{-1} \circ r_i(x_{n-1}))$. $\Box$

This theorem is similar to Theorem 9, relative to binary r-operators and their relations with s-operators

5.3 Termination properties of n-ary r-operators

We show in this section that any n-ary r-operator can be used in a *direct−association*. We need the following lemma.

Lemma 19 *Let P be a dipath in the digraph $G(V, E)$ modeling an associative net where the local computations on each computing entity v are based on an idempotent $(\delta^-(v)+1)$-ary r-operator built with the s-operator $\star$. Then its r-dipath-mapping r_P verifies $x \preceq_\star r_P(x)$.*

Proof. We show this result by recurrence on the length of the dipath P. It is true if $P = (v_0, v_1)$ since $r_{v_1}^{v_0}$ verifies $x \star r_{v_1}^{v_0}(x) = x$ (see Definition 12), which is equivalent to $x \preceq_\star r_{v_1}^{v_0}(x)$.

Suppose now the lemma is true for any dipath $P = (v_1, v_2) \dots (v_k, v_{k+1})$ of length k. Let v_0 be a direct ancestor of the vertex v_1 in G and P' the dipath obtained from P in adding the edge (v_0, v_1) at the beginning. Following recurrence hypothesis, we have $x \preceq_\star r_P(x)$ for all x in A, and in particular for $r_{v_1}^{v_0}(x) \in A$. We then have: $r_{v_1}^{v_0}(x) \preceq_\star r_P(r_{v_1}^{v_0}(x))$. Since for any x in A, $x \preceq_\star r_{v_1}^{v_0}(x)$ and $r_P(r_{v_1}^{v_0}(x)) = r_{P'}(x)$, we have $x \preceq_\star r_{P'}(x)$, that proves the lemma.

□

Theorem 20 *The $\diamond_G$−*direct−association* terminates on a deterministic result in any network using any data of A if the operator $\diamond_G$ is defined by: on the computing entity v of G, $\diamond_G = \diamond_v$, where $\diamond_v$ is an idempotent $(\delta^-(v)+1)$-ary r-operator based on the s-operator $\star$. Moreover, the result in each computing v is equal to:*

$$\star \left\{ r_P(b_0^0(u)), u \in \widehat{\Gamma_G^{-1}}(v), P \text{ elementary dipath from } u \text{ to } v \right\}$$

where r_P is the r-dipath-mapping associated to the dipath P.

Proof. There exists a date t such that any computing entity v in the network has received the initial datum of all its ancestor computing entities. That means that the expression in v at the date t is composed of r-mappings of the network, initial data and the operator $\star$. Since each r-mapping is an isomorphism of $(A, \star)$, it is possible to rewrite the expression in v according to the following transformation scheme (X_1, X_2, X_3 and X_4 can be identified to any sub-expressions):

$$X_1 \star r(X_2 \star X_3) \star X_4 = X_1 \star r(X_2) \star r(X_3) \star X_4$$

which gives on the expression $x \star r_1(y \star r_2(z \star r_3(t)))$ the equivalent expression $x \star r_1(y) \star r_1 \circ r_2(z) \star r_1 \circ r_2 \circ r_3(t)$. This process allows to recognize the r-dipath-mappings. Hence, if P is a dipath in G from the computing entity u (with the initial datum $b_0^0(u)$) to the computing entity v, the entity v will include $r_P(b_0^0(u))$ in its final expression (before any simplification in it).

If there is no cycle, the number of dipaths between any pair of vertices is finite and the result expression on v given by the $\diamond_G$−*direct−association* is $\star\{r_P(b_0^0(u)), u \in \widehat{\Gamma_G^{-1}}(v), P$ elementary dipath from u to $v\}$, which is a deterministic result.

Now suppose that there is a cycle C in G and let u and v be two vertices of this cycle. Then the number of dipaths (not elementary) from u to v is infinite: if P is an elementary dipath from u to v in G, then the dipaths PC, PCC, etc. are also dipaths (not elementary) from u to v. In that case, values built with the r-dipath-mappings r_P, r_{PC}, r_{PCC}, etc. will come to v. However, following Lemma 19, each r-dipath-mapping r_Q verifies, for all x in A, $x \preceq_\star r_Q(x)$. Then, for any $k \in \mathbb{N}$, for any dipath $PC^k = P\underbrace{C \dots C}_{k}$ from u to v, we have $r_{PC^k} = \underbrace{r_C \circ \dots \circ r_C}_{k} \circ r_P$ and, for any x in A, $r_p(x) \preceq_\star r_C(r_P(x)) \preceq_\star \dots \preceq_\star \underbrace{r_C \circ \dots \circ r_C}_{k}(r_P(x))$, which means that $r_P(x) \star \underbrace{r_C \circ \ldots \circ r_C}_{k}(r_P(x)) = r_P(x)$. Thus the data which come to v for second time modified by the r-dipath-mapping r_C of the cycle C are suppressed with the $\star$ computation on v and the result expression in v terminates on the finite length expression $\star\{r_P(b_0^0(u)), u \in \widehat{\Gamma_G^{-1}}(v), P$ elementary dipath from u to $v\}$. $\square$

6 Generalized r-operator

In this section, we study more general transformations on the incoming data than those previously studied. In *(Ducourthial, 1997)*, we introduced a unique transformation of all the incoming data towards the r-mapping. In the previous section, we extended this result to several r-mappings, one per incoming link. In this section, we study more general transformations. As we will see, more general transformations than isomorphisms of $(A, \star)$ lead to complex results that are difficult to use in practice. Next, the problem to know necessary conditions that insure termination of *direct−association*s in any network with any data will be discussed.

6.1 Definitions

First, we give a formal definition of the rs-mappings and r-mappings presented in Section 1.

Definition 21 *The operator $\diamond_v$ of Δ_v used for local computation on the computing entity v is said to be an* rs-mapping *if it is a mapping from $A^{\delta^-(v)+1}$ into A such that*
(1) *there exists an s-operator $\star$ on A,*
(2) *there exists an integer $l > 0$ and $l+1$ integers $0 < k_i \leq \delta^-(v)$, $(0 \leq i \leq l)$,*
(3) *there exists l mappings r_i from A^{k_i} into A, $(0 \leq i \leq l)$, called* r-mappings,

such that for any $\{x_0, \dots, x_{\delta^-(v)}\} \subset A$, $\diamond_v$ can be written as follows:

$$\diamond_v \left(x_0, \dots, x_{\delta^-(v)}\right) \quad = \quad r_0 \left(x_0^0, \dots, x_0^{k_0}\right) \star r_1 \left(x_1^0, \dots, x_1^{k_1}\right) \star \dots \star r_l \left(x_l^0, \dots, x_l^{k_l}\right)$$

where, for any p in $\{1, \dots, l\}$ and any q in $\{1, \dots, k_p\}$, x_p^q is in $\{x_0, \dots, x_{\delta^-(v)}\}$.

For example, if the in-degree of the computing entity v is $\delta^-(v) = 3$, and if $A = \mathbb{N}$, the following operator is an rs-mapping of $\Delta_v = \{\diamond : \mathbb{N}^4 \to \mathbb{N}\}$: $\diamond_v(x_0, x_1, x_2, x_3) = \mathbf{min}(x_0 - 2 \times x_1, (x_2+2) \times (x_3+2), 4 \times (x_1 + x_2 + x_3))$. In this example, the operator performs a maximum on the initial datum x_0 of v and three incoming data x_1, x_2 and x_3 after they have been modified with three r-mappings: the first one is $(x_0, x_1) \mapsto x_0 - 2 \times x_1$, the second one is $(x_2, x_3) \mapsto (x_2+2) \times (x_3+2)$ and the third one is $(x_1, x_2, x_3) \mapsto 4 \times (x_1 + x_2 + x_3)$.

Definition 22 *An rs-mapping $\diamond_v$ of Δ_v built with the s-operator $\star$,* preserves the initial datum *if there exists an rs-mapping $\diamond'_v$ of Δ_v such that, for any $(x_0, \dots, x_{\delta^-(v)})$ in $A^{\delta^-(v)+1}$, we have:*

$$\diamond_v \left(x_0, \dots, x_{\delta^-(v)-1}\right) = x_0 \star \diamond'_v \left(x_0, \dots, x_{\delta^-(v)-1}\right)$$

For instance, the following rs-mapping from $\Delta_v = \{\diamond : \mathbb{N}^4 \to \mathbb{N}\}$ preserves the initial datum: $\diamond_v(x_0, x_1, x_2, x_3) = \mathbf{min}(x_0, (x_2+2) \times (x_3+2), 4 \times (x_1 + x_2 + x_3))$.

Definition 23 *An r-mapping r from A^k into A of an rs-mapping built with the s-operator $\star$* increases all its variables *if for any $(x_0, \dots, x_{k-1})$ in A^k, we have:*

$$\forall i, 0 \leq i < k, \quad x_i \preceq_\star r\left(x_0, \dots, x_{k-1}\right)$$

As an example, the r-mappings $(x, y) \mapsto (x+2) \times (y+2)$ and $(x, y, z) \mapsto 4 \times (x+y+z)$ of the previous rs-mapping increase all their variables.

Definition 24 *An r-mapping r from A^k into A of an rs-mapping built with the s-operator $\star$ is $\star$-*compatible *if for any x_0, ..., x_{k-1} and y in A, for any i, $1 \leq i \leq k$, we have:*

$$r(x_1, \ldots, x_i \star y, \ldots, x_k) = r(x_1, \ldots, x_i, \ldots, x_k) \star r(x_1, \ldots, y, \ldots, x_k)$$

The r-mappings $(x, y) \mapsto r_1(x, y) = (x+2) \times (y+2)$ and $(x, y, z) \mapsto r_2(x, y, z) = 4 \times (x+y+z)$ are **min**-compatibles on $\mathbb{N}$: $r_1(\mathbf{min}(a, b), y) = (\mathbf{min}(a, b)+2) \times (y+2) = \mathbf{min}((a+2) \times (y+2), (b+2) \times (y+2)) = \mathbf{min}(r_1(a, y), r_1(b, y))$ and $r_2(\mathbf{min}(a, b), y, z) = 4 \times (\mathbf{min}(a, b)+y+z) = \mathbf{min}(4 \times (a+y+z), 4 \times (b+y+z)) = \mathbf{min}(r_2(a, y, z), r_2(b, y, z))$ (note that variables of r_1 and r_2 play similar roles).

We now define *generalized r-operator.*

Definition 25 *A* generalized n-ary r-operator *is an rs-mapping from A^n into A based on the s-operator $\star$ such that:*
(1) *it preserves the initial datum*
(2) *each of its r-mappings r_i is $\star$-*compatible *on A.*

The rs-mapping $\diamondsuit_v(x_0, x_1, x_2, x_3) = \mathbf{min}(x_0, (x_2+2) \times (x_3+2), 4 \times (x_1+x_2+x_3))$ base on **min**is a generalized 4-ary r-operator on $\mathbb{N}$. Indeed, it preserves the initial datum and its r-mappings are **min**-compatible. Idempotent generalized r-operators generalize the idempotent n-ary r-operators.

Definition 26 *A generalized r-operator is said idempotent if all its r-mappings increase all their variables.*

The generalized 4-ary r-operator $\diamondsuit_v(x_0, x_1, x_2, x_3) = \mathbf{min}(x_0, (x_2+2) \times (x_3+2), 4 \times (x_1 + x_2+x_3))$ is idempotent because its two r-mappings increase their variables.

Note that in a generalized idempotent r-operator, the first variable x_0 does not appear in any r-mapping. Indeed, since each r-mapping increases all its variables, an r-mapping r using x_0 is always greater than x_0. But since the generalized idempotent r-operator preserves the initial datum x_0, the r-mapping disappears in the $\star$ computation: $x_0 \star r(\ldots, x_0, \ldots) = x_0$

In an associative net where each computing entity uses an n-ary r-operator, we have seen that at each edge is associated an r-mapping. For generalized r-operators, it is not the case since a datum sent by a direct ancestor can be used by several r-mappings. For instance, in the previous rs-mapping $\diamondsuit_v(x_0, x_1, x_2, x_3) = \mathbf{min}(x_0, (x_2+2) \times (x_3+2), 4 \times (x_1+x_2+x_3))$, the datum x_2 sent by the second direct ancestor is used by both first and second r-mappings. We need to define mappings corresponding to each incoming edges, that we will call *r-edge-mappings.* These mappings cannot be defined without incoming data from other edges. For instance, in the previous example, the r-edge-mapping corresponding to the second incoming edge is $\mathbf{min}((x_2+2)\times(x_3+2), 4\times(x_1+x_2+x_3))$. It is a function of the variable x_2 that depends on the others data received by v from other direct ancestors (which are x_1 and x_3 in this example). For an r-edge-mapping r_v^u corresponding to the incoming edge (u,v) of the vertex v, we denote the other incoming data used in r_v^u by the wild-card x_v, and we call them *parametric variables*. The r-edge-mapping depending on the parametric variables x_v is denoted by $r^u_{v|x_v}$. For clarity, if unnecessary we do not define completely the parametric variables x_v. In that case we always suppose that they are data received by v to complete variables in the r-edge-mapping.

Definition 27 *Let v be a computing entity in an associative net such that the computing entities $u_1, \ldots, u_{\delta^-(v)}$ are the direct ancestors of v. Let $\diamond_v$ be an rs-mapping of Δ_v used by v for its local computations such that:*

$$\diamond_v\left(x_0, x_{u_1}\ldots, x_{u_{\delta^-(v)}}\right) \quad = \quad r_0\left(x_0^0,\ldots,x_0^{k_0}\right)\star r_1\left(x_1^0,\ldots,x_1^{k_1}\right)\star\cdots\star r_l\left(x_l^0,\ldots,x_{l^{k_l}}\right)$$

where, for any p in $\{1,\ldots,l\}$ and any q in $\{1,\ldots,k_p\}$, x_p^q is in $\{x_0, x_{u_1},\ldots,x_{u_{\delta^-(v)}}\}$.

We define the r-edge-mappings $r_v^{u_h}$*, for each h in $\{1,\ldots,\delta^-(v)\}$ by:*

$$r^{u_h}_{v|x_v}\left(x_{u_h}\right) = \mathop{\star}_{1\leq i\leq\delta^-(v)}\left\{r_i\left(x_i^1,\ldots,x_i^{k_i}\right), \exists j, 1\leq j\leq k_i, x_i^j = x_{u_h}\right\}$$

where x_v represents the parametric variables $x_i^j \neq x_{u_h}$.

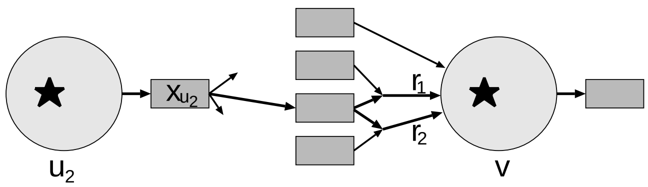

Figure 13: The r-edge-mapping $r^{u_h}_{v|x_v}$ is defined by $r^{u_h}_{v|x_{u_1},x_{u_2}}\left(x_{u_2}\right) = r_1(x_{u_1},x_{u_2})\star r_2(x_{u_2},x_{u_3})$.

Proposition 28 *Let v be a computing entity in an associative net such that v uses an rs-mapping $\diamond_v$ of Δ_v for its local computations. The following propositions are equivalent:*
(1) *Each r-mapping of the rs-mapping $\diamond_v$ increases all its variables,*
(2) *For each direct ancestor u of v, the r-edge-mapping $r^u_{v|x_v}$ increases its variable ($x \preceq_\star r^u_{v|x_v}(x)$ for all x in A).*

Proof. (1) $\Rightarrow$ (2) Let $r^u_{v|x_v}\left(x_u\right) = \mathop{\star}_{1\leq i\leq\delta^-(v)}\left\{r_i\left(x_i^1,\ldots,x_i^{k_i}\right), \exists j, 1\leq j\leq k_i, x_i^j = x_u\right\}$ be the r-edge-mapping corresponding to the edge (u,v). Since x_u is a variable used by all the r-mappings r_i of $r^u_{v|x_v}$, we have $x_u \preceq_\star r_i\left(x_i^1,\ldots,x_u,\ldots,x_i^{k_i}\right)$ for any r-mapping r_i of $r^u_{v|x_v}$. Thus we have: $x_u \preceq_\star \star\left\{r_i(x_i^1,\ldots,x_u,\ldots,x_i^{k_i}), r_i \text{ used in the definition of } r^u_{v|x_v}\right\}$, which means that $r^u_{v|x_v}$ increases its variable ($x_u \preceq_\star r^u_{v|x_v}(x)$).

(2) $\Rightarrow$ (1) Let x_u be a variable of an r-mapping r_i of the rs-mapping $\Diamond_v$ such that x_u is the variable sent by the direct ancestor u of v. Hence the r-mapping r_i is used to define the r-edge-mapping r_v^u corresponding to the edge (u, v). Since each r-edge-mapping increases its variable, we have: $x_u \preceq_\star r_v^u(x_u)$, which means that: $x_u \preceq_\star \{r_j(\ldots, x_u, \ldots)$, for any r-mapping r_j used to define $r_v^u\}$. Thus we have $x_u \preceq_\star r_i(\ldots, x_u, \ldots)$, that ends the proof. $\square$

In particular, this proposition states that in an associative net where each computing entity uses an idempotent generalized r-operator for its local computation, each r-edge-mapping increases its variable.

Hence, at each edge (u, v) of E, we can associate an r-edge-mapping. For an n-ary r-operator, it was simply the r-mapping associated to the edge. Using these r-mappings, we built r-dipath-mappings. We do the same with the r-edge-mappings.

Definition 29 *Consider an associative net modeled by the digraph $G(V, E)$ such that each computing entity v uses an rs-mapping for its local computations. For any dipath $P = (v_0, v_1) \ldots (v_{k-1}, v_k)$ in G, the r-dipath-mapping of P, denoted by $r_{P|x_{v_0},\ldots,x_{v_k}}$, is defined as follows:*

$$r_{P|x_{v_1},\ldots,x_{v_k}} \quad = \quad r^{v_{k-1}}_{v_k|x_{v_k}} \circ r^{v_{k-2}}_{v_{k-1}|x_{v_{k-1}}} \circ \ldots \circ r^{v_0}_{v_1|x_{v_1}}$$

One can see that, in an associative net where each computing entity uses a generalized r-operator based on the s-operator $\star$ for its local computation, any r-edge-mapping is $\star$-compatible and any r-dipath-mapping is $\star$-compatible too. This is due to the fact that the r-mappings of a generalized r-operator are $\star$-compatible. Moreover, Lemma 19 can be generalized to n-ary r-mappings as follows: any r-dipath-mapping increases its variable, because it is built from r-edges-mappings, which increase their variables.

6.2 Termination property of the generalized r-operators

As for n-ary r-operator, we show that generalized idempotent r-operators can be used for *direct−association*s in any network with any data of A. However, due to the generalization of the r-mappings, results are only recursively-defined. Lemma 30 gives a non-recursive result in the case the network is a dipath. Indeed, in this case, the r-edge-mappings have no parametric variables[9].

Lemma 30 (dipath) *Consider an associative net modeled by a digraph $G(V, E)$ which is a dipath, composed of vertices $v_0, \ldots, v_k$ and edges (v_i, v_{i+1}), $0 \le i < k$, and suppose that each computing entity v_i uses a generalized 2-ary r-operator based on the s-operator $\star$. Then the result R_k in v_k of a direct−association based on these operators in this network is:*

$$R_k \quad = \quad b_0^0(v_k) \quad \star \quad r_{v_k}^{v_{k-1}}\left(b_0^0(v_{k-1})\right) \quad \star \quad \ldots \quad \star \quad r_{v_k}^{v_{k-1}} \circ \ldots \circ r_{v_1}^{v_0}\left(b_0^0(v_0)\right)$$

[9]Note that in this case, the generalized r-operator does not need to be idempotent. This is due to the fact that there is no cycle in the network.

Proof. We prove this lemma by recurrence on the length of the dipath. It is true for $k=0$: the result in v_0 is $R_0 = b_0^0(v_0)$. Suppose now that the lemma is true until the length i: the result in v_i is $R_i = b_0^0(v_i) \star r_{v_i}^{v_{i-1}}(b_0^0(v_{i-1})) \star \ldots \star r_{v_i}^{v_{i-1}} \circ \ldots \circ r_{v_1}^{v_0}(b_0^0(v_0))$. Then the result in v_{i+1} is $R_{i+1} = b_0^0(v_{i+1}) \star r_1^{i+1}(R_i) \star \ldots \star r_k^{i+1}(R_i)$ where $r_1^{i+1}, \ldots, r_k^{i+1}$ are the r-mappings of the generalized binary r-operator used by the computing entity v_{i+1}. Since R_i is in each these r-mappings, they define the r-edge-mapping of the edge (v_i, v_{i+1}) and $R_{i+1} = b_0^0(v_{i+1}) \star r_{v_{i+1}}^{v_i}(R_i)$. Using the fact that the r-edge-mapping is $\star$-compatible, we can develop R_{i+1} and we obtain: $R_{i+1} = b_0^0(v_{i+1}) \star r_{v_{i+1}}^{v_i}(b_0^0(v_i)) \star \ldots \star r_{v_{i+1}}^{v_i}(r_{v_i}^{v_{i-1}} \circ \ldots \circ r_{v_1}^{v_0}(b_0^0(v_0)))$, that gives the lemma. □

The following lemma is needed to prove that any *direct−association* based on generalized r-operators terminates after a finite time, with any data on any network, and in particular on cycles. For this purpose, the operator must be idempotent, which means that all its r-mappings increase their variable, and so the r-edge-mappings and the r-dipath-mappings.

Lemma 31 (cycle) *Consider an associative net modeled by a digraph $G(V,E)$ where the local computations are based on generalized idempotent r-operators. Then, any r-dipath-mapping r_P where P is a cycle can be suppressed in the result expressions given by a direct−association.*

Proof. Note that only entities which are in a cycle, or which have an ancestor in a cycle, can have in their expression a r-dipath-mapping r_P where P is a cycle.

Let v be a computing entity in an associative net without any cycle. v will receive all the initial data of its ancestors, eventually modified by r-dipath-mappings. Moreover, all its descendants will receive the expressions it builds. Now suppose we add a link from v to one of its ancestors, that make a cycle P. The computing entity v will receive the expression that it has itself built inside the r-dipath-mapping r_P, and this expression will be inserted into its own. But using the $\star$-compatibility of the r-dipath-mapping, what received by v with r_P can be developed, and using the fact that r_P increases its variable, these new terms can be suppressed because they are already in the expression of v. Descendants of v will then never receive a result containing r_P. □

The following lemma gives the result of a *direct–association* based on generalized r-operators in the case the network is a dag[10]. Note that this lemma is not so strong and is easy to prove. Indeed, the result is recursive: to obtain the result R_v in a computing entity v, the result of all the ancestors of v are needed. It is due to the fact that the r-mappings of several variables considerably complicate the expression of the result, except in trivial cases as in Lemma 30.

Lemma 32 (dag) *Consider an associative net modeled by the digraph $G(V,E)$ where G is a dag, and where each computing entity v uses a generalized $(\delta^-(v)+1)$-ary r-operator. Then, the result R_w of any computing entity w in G is*

$$R_w \quad = \quad b_0^0(w) \star \left\{ r^u_{w|R_v, v \in \Gamma_G^{-1}(w) \setminus \{u\}}(R_u), u \in \Gamma_G^{-1}(w) \right\}$$

[10]As for Lemma 30, since their is no cycle in the digraph, the idempotency property is no needed.

Finally, the two preceding lemma allow to conclude on the termination of a ***direct–association*** based on generalized idempotent r-operators in any network and with any data. However, as for dags, the result is recursively-defined, and the theorem is not strong, nor difficult to prove. Lemma 31 states that this recursive definition is well defined, even if there is some cycle. The theorem is obtained by considering, one at a time, each vertex v as it was the root of a dag. In this case, result is obtained from Lemma 32. Now, if some cycles are added, they do not change results on v, as stated by Lemma 31. Finally, the $\star$-compatibility property of the r-edge-mappings allows to develop any results, and then to obtain an expression composed of initial data, r-edge-mappings and the s-operator $\star$. The properties of the $\star$ operator indicates that the order of the data in each computing entity is not important, which means that the result is deterministic.

Theorem 33 *The $\diamond_G$–**direct–association** will terminate on a deterministic result in any network using any data of A if the operator $\diamond_G$ is defined by: on the computing entity v of G, $\diamond_G = \diamond_v$, where $\diamond_v$ is an idempotent generalized $(\delta^-(v)+1)$-ary r-operator based on the s-operator $\star$. Moreover, the result R_v in each computing entity v is equal to:*

$$R_v \quad = \quad b_0^0(w) \star \left\{ r^u_{w|R_v, v \in \Gamma_G^{-1}(w) \setminus \{u\}}(R_u), u \in \Gamma_G^{-1}(w) \right\}$$

6.3 Remarks

A recursively defined result is not easy to use for programming because it could be difficult to recognize what result has been built by the operator.

There exists some cases where the result is known under a more practice form. That appears when the r-mappings are "separable" in $(A, \star)$:

Definition 34 *An r-mapping r from A^k into A is* separable *if there exists k mappings $r_1, \dots,$ r_k from A to A so that for any $x_1, \dots, x_k$ in A, $r(x_1, \dots, x_k) = r_1(x_1) \star \dots \star r_k(x_k)$.*

A non recursive result can be obtained with generalized r-operators having all their r-mappings separable. Indeed, in that case there is no more parametric variables in the definition of the r-edges mappings and compositions of these r-edges mappings can be expressed as r-dipath mappings, as with n-ary r-operators. One can also remark that in this case, the associative net can be modified in adding some new nodes in such a way that there is one and only one r-mapping per edge.

However, the main difference between this subclass of the generalized r-operators and the n-ary r-operators is that the r-dipath mappings are not necessarily bijective mappings. Due to the $\star$-compatibility, the r-mappings are homomorphisms of $(A, \star)$. As an example, the rs-mapping $\diamond_v(x_0, x_1, x_2) = \mathbf{min}(x_0, 2 \times x_1, 4 \times x_2)$ is not an n-ary r-operator on $\mathbb{N}$ because its r-mappings are not bijective on $\mathbb{N}$. However, it is a generalized idempotent 3-ary r-operator on $\mathbb{N}$ and any ***direct – association*** that uses such an operator terminates on any network G, with any data, and gives the following result on each computing entity v:

$$\star \left\{ r_P(b_0^0(u)), u \in \widehat{\Gamma_G^{-1}}(v), P \text{ elementary dipath from } u \text{ to } v \right\}$$

Finally, some generalized r-operators with unseparable r-mappings of several variables have a result which can be used in programming, even if Theorem 33 predict a recursively defined result (see Section 7).

6.4 Discussion

The r-operators have been built from s-operators by adding modifications of all data sent by the direct ancestors. We called this property *preserving the initial datum* (Definition 22).

One can show that preserving the initial datum is necessary. Indeed, consider the network displayed in Figure 14. Suppose that the computing entity computes a minimum and that r does not preserve the initial datum. In that case, the successive expressions built by the entity are $[r(x_0)]$, $[r^2(x_0)]$, $[r^3(x_0)]$ and so on. Hence, in the general case (no special hypothesis on r or x_0), the computation does not terminate.

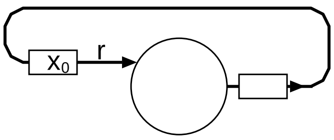

Figure 14: The initial datum must be preserved.

Another hypothesis we used was the fact that any modification of an incoming datum always produces a greater datum. We said that the r-edge mapping *increases its variable* (Definition 23). This property was given by the idempotency property of the binary and n-ary r-operators: $x \triangleleft x = x \Leftrightarrow x \star r(x) = x \Leftrightarrow x \preceq_\star r(x)$. This property was extended to generalized r-operators: a generalized r-operator is idempotent if all its r-mappings increase their variable (Definition 26), which is equivalent to the fact that any r-edge-mapping increases its variable (Proposition 28).

One can show that this property is necessary. Indeed, consider the network of Figure 15 and suppose that the computing entity computes a minimum. If r does not increase its variable, then after a moment, the result of the minimum computation will always be the datum received on the second incoming link and modified by r, and never x_0. Hence in general the calculation does not terminate.

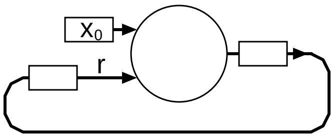

Figure 15: r must increase its variable.

We would like to find conditions on operators, independently of the network. However, it is sufficient to only suppose that r-dipath-mappings r_P increase their variable if P is a cycle. But this condition concerns both the operators and the network, that we would not.

Finally, to recognize the result and to assure it is deterministic, we supposed some properties on the r-mappings. Hence, we used isomorphisms for the binary and n-ary r-operators. Isomorphisms allowed to develop expressions and then to recognize iterations of r for binary r-operators (r^{distance}), or r-dipath-mappings for n-ary r-operators. This property is extended to generalized r-operators with the $\star$-compatibility (Definition 24).

7 Applications

7.1 Introduction

Theorem 10 indicates that a *direct–association* with binary idempotent r-operators gives results that contain the distance information, while only the strongly connected component related information is contained in results given by a *direct–association* with s-operators (see Theorem 4). Thus, these operators make better use of the information coded by the network itself than the s-operators. In *(Ducourthial, 1997)*, we gave an application of this property: the distance computation.

Theorem 20 states that any n-ary r-operator includes the dipaths information in the results it builds. This graph information is more complete than the distance information since it is possible to obtain the distance information from the dipaths information but the converse is false.

Hence, to make benefit of the graph information in an associative net, the n-ary r-operators are more suited than the binary ones, which are more suited than the s-operators. Note that an s-operator is an idempotent r-operator with the r-mapping identity $\mathbf{Id} : x \mapsto x$. Moreover, a binary (idempotent) r-operator based on the r-mapping r is an (idempotent) n-ary r-operator with its $n-1$ r-mappings equal to r. This class of operators which it is now possible to use for the *direct – association*s is larger and the new operators allow to make better use of the information coded by the network.

7.2 Examples of problems solved with r-operators

In this section, we briefly give some examples of r-operators designed for solving particular problems.

7.2.1 Distance computation, shortest dipath spanning tree and forest

The distance computation from a main source in a digraph is solved with the binary r-operator **minc** defined by $\mathbf{minc}(x, y) = \mathbf{min}(x, y+1)$ on $\overline{\mathbb{Z}} = \mathbb{Z} \cup \{+\infty\}$.

After the **minc**–*direct–association*, if each entity choses one of its incoming buffer containing the smallest received datum, a shortest dipath spanning tree rooted on the main source is obtained.

If there is several (main) sources in such a way each entity has a source in its ancestors, each node will be in the tree which has the closest root from it and the shortest dipath spanning forest is solved with a single *direct – association*.

7.2.2 Single and multiple source shortest path problem

This problem is similar to the distance computation, except that the edeges have a weight not always equal to 1. A $\mathbf{minc}_w$ − *direct* − *association* solves the problem, where $\mathbf{minc}_w$ is an n-ary r-operator on $\overline{\mathbb{Z}}$ defined on each entity by:

$$\mathbf{minc}_w\,(x_0, \ldots, x_{n-1}) = \mathbf{min}\,(x_0, x_0 + w_1, \ldots, x_{n-1} + w_{n-1})$$

Each w_i is the weight of the i^{th} incoming edge of the entity.

As for the shortest dipath spanning forest, the same operator solves in a single *direct* − *association* the multiple source shortest path problem in the case there are several (main) sources, and each entity has one source in its ancestors.

7.2.3 Depth-first-search tree

For this algorithm, we need the identifiers of the computing entities and we suppose there exists a total order on these identifiers. For instance, when the identifiers are integers, the order is given by the natural order on the integers. Such an order relation can be arbitrary created if necessary. We considere the set A of the ordered lists of identifiers and the s-operators $\star$ which gives the smallest list using the lexicographic order on the lists of identifiers. For instance, if the identifiers are letters $a, b, c, \ldots$ ordered by the alphabetical order, the operator $\star$ gives the following results:

$$\begin{array}{lcl} (a, b, d, e,) \star (a, b, c, d, e, f) & = & (a, b, c, d, e, f) \\ (a, a) \star (a) & = & (a) \\ (f, g, f) \star (f, f) & = & (f, f) \end{array}$$

More, we considere on each node v the r-mapping

$$\begin{array}{lcl} A & \longrightarrow & A \\ l & \mapsto & r(l) = l \cup (v) \end{array}$$

where $\cup$ is the concatenation of two lists. It is easy to verify that $l \star r(l) = l$ and then the r-operator $l \lhd l' = l \star r(l')$ is idempotent.

So any *direct* − *association* based on this operator will terminate. If each computing entity chooses the unique direct-ancestor that sent to it the smallest list, we then obtain a depth-first-search tree. Next, an integer numbering of the vertices of the network can simply be obtained by any injective mapping from A into $\mathbb{N}$ since A is totally ordered by $\star$.

7.2.4 Diameter

For this problem, we consider the set A of the lists of sets of identifiers of the computing entities, such as, for instance: $(\{d\}, \{b\}, \{a, c\})$, where a, b, c, and d are identifiers of computing entities.

The s-operator $\star$ is the fusion term-to-term of two lists, deleting vertices that appear several times in the list, but the first occurence. For instance:

$$(\{d\},\{b\},\{a,c\}) \star (\{c\},\{a,e\},\{b\}) = (\{d,c\},\{b,a,e\},\{\cancel{a},\cancel{e},\cancel{b}\}) = (\{d,c\},\{b,a,e\})$$

It is easy to verify that it is an s-operator. We identify any list with empty sets at the end to the same list without empty sets: $(\{d\},\{b\},\{a,c\}) = (\{d\},\{b\},\{a,c\},\emptyset,\dots,\emptyset)$.

The unique r-mapping that we consider is:

$$\begin{array}{lll} A & \longrightarrow & A \\ l & \mapsto & r(l) = \{\emptyset\} \cup l \end{array}$$

where $\cup$ is the concatenation of two lists. For instance, $r\,(\{d\},\{b\},\{a,c\}) = (\emptyset,\{d\},\{b\},\{a,c\})$. r is not a bijection, then we have not a binary r-operator but a generalized one. We then define the generalized operator $\diamond_v$ by:

$$\diamond(l_0,\dots,l_{\delta^-(v)-1}) = l_0 \star r(l_1) \star \dots \star r(l_{\delta^-(v)-1})$$

$\diamond_v$ preserves the initial datum. Moreover r increases its variable since $r(l) \star l = l$, which means that $l \preceq_\star r(l)$, and it is an homomorphism of $(A,\star)$ and *a fortiori* $\star$-compatible: $r(l \star l') = r(l) \star r(l')$.

Thus the *direct−association* based on this operator terminates. Since the r-mapping r is a mapping with a single variable, the result is known in a non-recursive form, as for binary r-operator. When each computing entity u is initialized with the list $(\{v\})$, the result in v is:

$$\star\left\{ r^{\mathrm{d}_G(u,v)}(\{u\}),\, u \in \widehat{\Gamma_G^{-1}}(v) \right\}$$

which is a complete representation of the ancestors of v, ordered by the distance to v: the first element of the list is the set containing v itself, the second is the set of the direct ancestors, and so one until the last ancestors.

The larger list gives the diameter of the digraph. The lengths of the lists can be computed localy and they can be diffused with a **max**−*direct−association*. Each entity obtains then the diameter of the digraph. Notes that the operator $\diamond_v$ gives more informations than the diameter knowledge.

7.2.5 Mathematical morphology

Opening by reconstruction is a powerful tool of mathematical morphology whose purpose is to suppress noise and generate constant grey level values, while preserving the contour information. Hence it is widely used as preprocessing step for image segmentation. For short, this image analysis operation can be assimilated, for each computin entity v (*i.e.* each pixel), to a maximum computation on the direct ancestors (neighbor's pixels), followed by a minimum computation with value of v (the pixel on v). It is then an rs-mapping $\diamond_v$ defined on $\overline{\mathbb{N}} = \mathbb{N} \cup \{+\infty\}$ by (in a 8-connected image[11]):

$$\diamond_v(p_0,p_1,\dots,p_8) = \mathbf{min}\,(p_0,\mathbf{max}(p_1,\dots,p_8))$$

[11] Each pixel has eight neighbors.

The initial datum is preserved, the r-mapping $(p_1, \dots, p_8) \mapsto \mathbf{max}(p_1, \dots, p_8)$ increases all its variables (in sense of the total ordering induced by **min**, and it is **min**-compatible. Hence the rs-mapping $\Diamond_v$ is a generalized idempotent r-operator.

8 Conclusion

In this report, we extended the definition of the associaitve nets to allow the use of n-ary operators. Next, we proposed a new class of operators that insure termination of *direct−association* on any network with any data: the n-ary r-operators. This class is larger than the one of r-operators, which is larger than the one of the s-operators. The n-ary r-operators have interesting properties which can be used in many applications.

Some weaker hypothesis can be made on operators, that still lead to the termination of *direct−association*. Hence we defined generalized r-operators.

All these operators have interesting properties for solving various problems such as distance computation, single and multiple source shortest path problem, depth-first-search tree, diameter or some operators of mathematical morphology.

Our future works are in the study of self-stabilizing properties of r-operators.

References

Chang, E. J. H. (1982). Echo algorithms: Depth parallel operations on general graphs. *IEEE Transactions on Software Engineering*, SE 8(4):391–401.

Dijkstra, E. W. and Scholten, C. S. (1980). Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4.

Ducourthial, B. (1995). Etude du modèle réseau associatif. Master's thesis, Laboratoire de Recherche en Informatique, bât. 490, Université Paris-Sud, Orsay, France. *(Mémoire de DEA Architecture Parallèle)*.

Ducourthial, B. (1997). Properties of the r-operators to insure termination of distributed computations. Technical Report 1140, Laboratoire de Recherche en Informatique, Bât. 490. Université Paris-Sud, 91405 Orsay, France.

Ducourthial, B. (to appear). Demi-vagues utilisant des r-opérateurs : applications et terminaison. Actes des Dixièmes Rencontres Francophones du Parallélisme, Strasbourg, France (RenPar'10).

Dulac, D. (1996). *Contribution au Parallélisme Massif en Analyse d'Images : Une Architecture SIMD Fondée sur la Reconfigurabilité et l'Asynchronisme*. PhD thesis, Institut d'Electronique Fondamentale, Université Paris-Sud, 91405 Orsay, France.

Hélary, J.-M., Maddi, A., Plouzeau, N., and Raynal, R. (1987). Parcours et apprentissage dans un réseau de processus communicants. *Techniques et Sciences Informatiques*, 6(2):127–139.

Mérigot, A. (1992). Associative nets: A new parallel computing model. Technical Report 92-02, Institut d'Electronique Fondamentale, Université Paris-Sud, 91405 Orsay, France.

Mérigot, A. (1997). Associative nets: A graph-based parallel computing net. *IEEE Transactions on Computers*, 46(5):558–571.

Raynal, M. and Hélary, J.-M. (1988). *Synchronisation et contrôle des systèmes et des programmes répartis*. Eyrolles, Paris, France.

Segall, A. (1983). Distributed network protocols. *IEEE Transactions on Information Theory*, IT 29(1):23–35.

Tel, G. (1987). Directed network protocols. In van Leeuwen, J., editor, *Distributed Algorithm. Second International Workshop*, volume 312 of *Lecture Notes in Computer Science*, pages 13–29, Amsterdam, The Netherlands. Springer Verlag.

Tel, G. (1991). *Topics in Distributed Algorithms*, volume 1 of *Cambridge International Series on Parallel Computation*. Cambridge University Press, Cambridge, UK.

Tel, G. (1994). *Introduction to Distributed Algorithms*. Cambridge University Press, Cambridge, UK.