On fairness in distributed automated deduction *

Maria Paola Bonacina **Jieh Hsiang**
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400, USA
{bonacina,hsiang}@sbcs.sunysb.edu

Abstract

We present a new approach to distributed automated deduction and in this context we propose solutions to the problem of *distributed uniform fairness*. Given a sequential theorem proving strategy and an input problem, our methodology partitions the clauses and the expansion inferences among the nodes of the distributed environment, establishing a derivation at each node. A distributed derivation is defined as a family of concurrent, asynchronous derivations, communicating clauses through message-passing. As soon as one of the local derivations finds a proof, the distributed derivation halts successfully.

Uniform fairness requires that all clauses which can be derived from persistent, non-redundant parents are generated eventually. In the distributed case, it comprises fairness of message-passing, which is necessary to make sure that any two persistent, non-redundant clauses, residents at different nodes, meet eventually. We reduce the abstract definition of uniform fairness for general derivations to a more concrete specification of uniform fairness for distributed derivations, by proving that the latter implies the former. Then, we describe mechanisms which can be embedded in a strategy to satisfy these conditions.

We conclude the paper with some discussion on the relevance of our results to other applications of distributed data bases.

1 Introduction

In this paper we present a new approach to distributed automated deduction called *Clause-Diffusion*; in this context we describe the associated notion of *distributed fairness* of a derivation and propose solutions to the problems involved.

A theorem proving strategy consists of an *inference mechanism* (a set of inference rules) and a *search plan*. We further refine the inference mechanism into *expansion* inference rules, e.g. resolution and paramodulation, and *contraction* inference rules, e.g. subsumption and simplification via rewrite rules. The motivation for studying distributed theorem proving is to improve the efficiency of theorem proving strategies by exploiting parallelism. The Clause-Diffusion approach is concerned mainly with *parallelism at the search level*, by partitioning the search space among

*Research supported in part by grant CCR-8901322, funded by the National Science Foundation. The first author is also supported by a scholarship of Università degli Studi di Milano, Italy.

concurrent, asynchronous theorem proving processes. These processes traverse their portions of the search space of the given problem, looking for a solution. As soon as one of them succeeds, the entire process succeeds and terminates. Since each process is assigned only a segment of the data, the processes need to communicate so that a proof involving data at different processes can be found. The processes send their clauses to other processes in form of messages, termed *inference messages*. Here we encounter the problem of *fairness*, as a requirement that the policies which control the handling of inference messages need to satisfy.

Fairness is a property of the search plan of a theorem proving strategy [5]. Intuitively, a fair search strategy ensures that no inference step which is necessary to find a proof will be postponed forever. Fairness is guaranteed by a stronger condition, *uniform fairness*, which requires that expansion steps between persistent clauses, those that are not deleted by contraction, are considered. Since two persistent clauses may be stored at two different nodes in a distributed environment, distributed uniform fairness requires that any two such clauses meet eventually at some site. Thus, it poses an additional fairness requirement on the communication part of a distributed theorem strategy, i.e. the part of the search plan which establishes when to send messages and when and how to process received messages.

The first result concerning fairness in the paper is to turn these intuitions into formal requirements, to be realizable by actual procedures. We start by giving a definition of *distributed derivations* and extend the definition of uniform fairness of a sequential derivation to a distributed derivation. If only one theorem proving process is active, then distributed derivation and distributed uniform fairness reduce to their sequential counterparts. We then present a set of more concrete conditions and prove that they are *sufficient* for the uniform fairness of a distributed derivation. To our knowledge, this is the first analysis of fairness in distributed deduction. Several techniques for implementing these sufficient conditions for uniform fairness in different architectures are described. These techniques introduce a certain amount of additional *redundancy*. We present a new contraction inference rule to delete redundant messages. These techniques and inference rule are implemented in our distributed theorem prover *Aquarius* [7].

Our Clause-Diffusion methodology for distributed theorem proving and study of distributed uniform fairness apply to theorem proving in general. However, we shall emphasize on *contraction-based strategies*. The distinguishing features of these strategies are a well-founded ordering $\succ$ on terms, equations and clauses, powerful contraction rules, strong restrictions to the application of expansion rules and a *simplification-first* search plan [12], i.e. a search plan which gives priority to contraction steps. The motivation for emphasizing simplification-based strategies is two-fold. First, there are both experimental evidence and theoretical understanding that they are more effective than expansion-oriented methods. Second, the issues of parallelism and fairness are more difficult and more interesting for contraction-based strategies than for other strategies, due to the dynamic behaviour of the data base caused by contraction. In this sense, strategies without contraction can be regarded as a special case of the study of strategies with contraction.

The rest of the paper is organized as follows. We first recall the basic definitions in theorem proving, including the definition of uniform fairness of a sequential derivation. We then outline the Clause-Diffusion methodology for distributed deduction. The treatment of fairness follows: formal definition, reduction to concrete requirements and techniques to implement them. We

conclude the paper with some discussion on the relationships between our work and the general issue of fairness in applications of distributed data bases.

2 Basic concepts in contraction-based deduction

A *theorem proving problem* consists in deciding, given a set of clauses S and a clause φ, whether φ is a theorem of S. A theorem proving strategy $\mathcal{C}$ is specified by a set of *inference rules* I and a *search plan* Σ. *Expansion* inference rules derive new clauses from existing ones, and add them to the data base. Resolution, hyperresolution and paramodulation are examples of expansion rules. *Contraction* inference rules delete existing clauses or replace them by logically equivalent but smaller ones. Examples of contraction rules are (proper) subsumption [17], simplification [20], tautology elimination, conditional simplification and normalization.

The search plan Σ chooses the inference rule and the premises for the next step. By iterating the application of I and Σ, a *derivation*

$S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \ldots S_i \vdash_{\mathcal{C}} \ldots,$

is constructed. A derivation is *successful* if it reaches a solution. A theorem proving strategy $\mathcal{C}$ is *complete*, if, whenever the input target is indeed a theorem, the derivation constructed by $\mathcal{C}$ halts successfully. Completeness involves both the inference rules I and the search plan Σ. First, it requires that if the input target is a theorem, there exist successful derivations by I (*completeness of the inference mechanism*). Second, it requires that whenever successful derivations exist, the search plan Σ selects a successful derivation among the possible derivations by I from the given input (*fairness of the search plan* [1]).

Fairness for theorem proving is implied by a stronger fairness property, which we call *uniform fairness* (e.g. [2, 3, 4, 13, 20]). We adopt here the definition given in [4]. This definition uses two additional concepts: *redundant* clauses and *persistent* clauses. Intuitively, a clause is *redundant* in a derivation if it is not necessary to prove the given target theorem [4, 6, 20]. Approaches to capture the notion of redundancy usually assume the existence of a *well-founded ordering on the proof structure* [2, 3, 4, 6, 12, 20, 22]. Given such an ordering, redundant data are identified as those whose deletion does not increase the complexity of proofs. Contraction inference rules are designed as concrete mechanisms to delete redundant data.

The notion of persistent clauses appeared first in [3, 13]: a clause is *persistent* in a derivation if it is generated at some stage of the derivation and never deleted afterwards. Given a derivation starting from a presentation S_0, the possibly infinite set $S_\infty = \bigcup_{j\geq 0} \bigcap_{i\geq j} S_i$ of all the persistent clauses is called the *limit* of the derivation. Given a strategy $\mathcal{C} =< I; \Sigma >$ and a set of clauses S, we denote by $I_e(S)$ the set of clauses which can be generated from S in one step by an expansion rule of I. We denote by R the *redundancy criterion* associated to $\mathcal{C}$ [4]. For a set of clauses S, $R(S)$ is the set of all the clauses which are redundant in S based on R. The redundancy criterion R is associated to $\mathcal{C}$ in the sense that whenever a contraction rule of I deletes or replaces a clause ψ in S, ψ is in $R(S)$. A redundancy criterion is required to be *monotonic*, i.e. if $S_1 \subseteq S_2$, then

[1] The use of the word "fairness" rather than "completeness" for the search plan has been inspired by its use for Knuth-Bendix type completion procedures.

$R(S_1) \subseteq R(S_2)$. Other requirements may be found in [4]. Finally, uniform fairness says that *all clauses that can be derived from persistent, non-redundant clauses should be generated eventually*:

Definition 2.1 (Bachmair and Ganzinger 1992) [4] *A derivation*

$S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \ldots S_i \vdash_{\mathcal{C}} \ldots$

by a strategy $\mathcal{C}$ *is* uniformly fair *if* $I_e(S_\infty - R(S_\infty)) \subseteq \bigcup_{j \geq 0} S_j$.

Studies of contraction and redundancy are motivated by the observation that contraction inference rules are an indispensable part of many successful theorem provers, e.g. [1, 15, 19]. From the theoretical side, however, their inclusion had posed a great challenge to the completeness proofs of theorem proving strategies, because in the presence of contraction steps the data base of clauses is not monotonically increasing during a derivation. It is only until recently that formal techniques for dealing with the completeness issues of contraction rules have been discovered [2, 6, 20, 4].

3 Distributed deduction

Given a complete theorem proving strategy $\mathcal{C} =< I; \Sigma >$, we address the problem of how to execute $\mathcal{C}$ in a distributed environment. By a *distributed environment* we mean a network of computers or a loosely coupled, asynchronous multiprocessor with distributed memory. The latter may be endowed with a shared memory component. Our "Clause-Diffusion" methodology does not depend on a specific architecture; it can be realized on different ones. Parameters such as the amount of *memory at each processor*, the availability of *shared memory* and the *topology* of the interconnection of the processors or *nodes*, are variable.

The basic idea in our approach is *to partition the search space* among the nodes. The search space is determined by the input clauses and the inference rules. At the clauses level, the input and the generated clauses are distributed among the nodes. For this purpose we need an *allocation algorithm*, which decides where to allocate a clause. Once a clause ψ is assigned to processor p_i, ψ becomes a *resident* of p_i. In this way each node p_i is allotted a subset S^i of the global data base. The union of all the S^i's, which are not necessarily disjoint, forms the current *global data base* S. Each processor is responsible for applying the inference rules in I to its residents, according to the search plan Σ. Since the global data base is partitioned among the nodes, no node is guaranteed to find a proof using only its own residents. To ensure that a solution will be found when one exists, the nodes need to exchange information, by sending each other their residents in form of messages, called *inference messages*. The inference messages issued by p_i let the other processors know which clauses belong to p_i, so that they can use them to perform inferences with their own residents. In a purely distributed system, the inference messages may be sent via routing or broadcasting. If a mixed environment, i.e. with distributed memory and a shared memory component, they may be communicated through the shared memory.

The separation of residents and inference messages can also be used to partition the search space at the inference level. Using the paramodulation inference rule as an example, one may establish that the inference messages are paramodulated *into* the residents, but not vice versa. This restriction has two purposes. First, it distributes the expansion inference steps among the

nodes. Second, it prevents a systematic duplication of steps: if this restriction were not in place, then paramodulation steps between two residents ψ_1 of p_1 and ψ_2 of of p_2 would be performed twice, once when ψ_1 visits p_2 and once when ψ_2 visits p_1. Other expansion inference rules can be treated in a similar way. While subdividing the expansion steps serves its purpose, it is not productive to subdivide the contraction steps, since the motivation behind contraction is to keep the data base always at the minimal. In a *contraction-based strategy*, an expansion step should be performed only if all the premises are fully reduced, at least with respect to the local data base. To ensure this, we require that each processor keep both its residents and received inference messages fully contracted.

Let us call the clause newly generated from an expansion step a *raw clause*. In the presence of contraction rules, a raw clause does not become a resident until it has been fully contracted. Thus, our method also features a number of *distributed contraction schemes* [7] to reduce a raw clause with respect to the global data base. After contraction, a raw clause becomes a *new settler*. New settlers are given to the allocation algorithm to be assigned to some node. Remark that we do not assume a central control process devoted to execute the allocation algorithm. Every process executes the allocation algorithm for its new settlers: it may decide either to retain a new settler or to send it to another node. The purpose of the allocation algorithm is to partition the search space and keep the work-load balanced as much as possible.

This is the basic working of the "Clause-Diffusion" approach to distributed automated deduction: inter-contraction and local expansion inferences at the nodes among residents and inference messages, distributed contraction of raw clauses, allocation of new settlers, and mechanisms for passing inference messages. By specifying the inference mechanism I, the search plan Σ to schedule inference steps and communication steps, the allocation algorithm, the distributed contraction scheme and the algorithms for routing and broadcasting of messages, one obtains a specific strategy. We refer to [7] for full detail of the methodology and its implementation.

The above elements are summarized in the following notion of *distributed derivation*: every processor p_k, $1 \leq k \leq n$, computes a derivation

$$(S; M; CP; NS)_0^k \vdash_{\mathcal{C}} (S; M; CP; NS)_1^k \vdash_{\mathcal{C}} \ldots (S; M; CP; NS)_i^k \vdash_{\mathcal{C}} \ldots$$

where S_i^k is the set of *residents*, M_i^k is the set of *inference messages*, CP_i^k is the set of *raw clauses* and NS_i^k is the set of *new settlers* at p_k at stage i.

A distributed derivation is the collection of the asynchronous derivations computed by the nodes. The *state* of the derivations at processor p_k and stage i is represented by the tuple $(S; M; CP; NS)_i^k$. More components may be added if indicated by a specific strategy. A distributed derivation succeeds as soon as the derivation at one node finds a proof. A step in a distributed derivation can be either an *expansion* step or a *contraction* step or a *communication* step. For instance, sending an inference message for $\psi \in S^k$ from node p_k to an adjacent node p_j can be written as $(S^k \cup \{\psi\}, M^j) \vdash (S^k \cup \{\psi\}, M^j \cup \{\psi\})$. Settling a new settler at node p_k can be written as $(S^k, NS^k \cup \{\psi\}) \vdash (S^k \cup \{\psi\}, NS^k)$. This representation assumes that communication between any two adjacent nodes is instantaneous. It does *not* assume, however, that communication between *any* two nodes is instantaneous. If an inference message sent by p_i reaches p_j through $p_{x_1} \ldots p_{x_m}$, it appears first in M^{x_1}, then in M^{x_2} and so on. The time elapsed

in going from the source to the destination is captured in our description, by showing the message stored, at successive stages, in the appropriate component of all the nodes on the path.

4 Uniform fairness of distributed derivations

In order to extend Definition 2.1 to the distributed case, we need to define the limit of a distributed derivation. First, we define the *local data base* at node p_k at stage i as the union $G_i^k = S_i^k \cup M_i^k \cup CP_i^k \cup NS_i^k$. Then, the *local limit* at processor p_k is $G_\infty^k = \bigcup_{i\geq 0} \bigcap_{j\geq i} G_j^k$. The *global data base* at stage i is the union of the local data bases, i.e. $G_i = \bigcup_{k=1}^n G_i^k$, and the *global limit* is $G_\infty = \bigcup_{k=1}^n G_\infty^k$. Local and global limits may be defined similarly for each component of the states in a distributed derivation, e.g. S_∞^k and S_∞, M_∞^k and M_∞. Then, the definition of uniform fairness is extended to a distributed derivation as follows:

Definition 4.1 *A distributed derivation*

$(S; M; CP; NS)_0^k \vdash_{\mathcal{C}} (S; M; CP; NS)_1^k \vdash_{\mathcal{C}} \ldots (S; M; CP; NS)_i^k \vdash_{\mathcal{C}} \ldots,$

for all k, $1 \leq k \leq n$, *is* uniformly fair *if* $I_e(G_\infty - R(G_\infty)) \subseteq \bigcup_{k=1}^n \bigcup_{j\geq 0} G_j^k$.

The following three conditions form a more concrete specification of uniform fairness of distributed derivations.

1. All messages should be processed eventually and thus there are *no persistent messages*:

   $\forall k,\ 1 \leq k \leq n,\ M_\infty^k = CP_\infty^k = NS_\infty^k = \emptyset$.

2. Given k and $\psi \in S_\infty^k$, we define the *abstract birth-time* of ψ in k to be the smallest index $i \geq 0$ such that $\psi \in \bigcap_{j\geq i} S_j^k$.[2] Then for every node p_k and for every persistent, non-redundant resident φ at p_k, *all persistent, non-redundant residents at the other nodes will eventually appear as inference messages at* p_k, *after the birth of* φ:

   $\forall k,\ 1 \leq k \leq n,\ \forall \varphi \in (S_\infty^k - R(S_\infty^k))$, if i is the abstract birth-time of φ, then $\forall h,\ 1 \leq h \neq k \leq n,\ \forall \psi \in (S_\infty^h - R(S_\infty^h))$, there exists an $l > i$ such that $\psi \in M_l^k$.

   Notice that i and l are stages of the same derivation, i.e. the derivation at p_k.

3. The derivation is uniformly fair with respect to the local inferences at each node. That is, every clause that can be generated from persistent, non-redundant clauses at p_k will be generated: $\forall k,\ 1 \leq k \leq n,\ I_e(S_\infty^k - R(S_\infty^k)) \subseteq \bigcup_{i\geq 0} CP_i^k$.

While Condition 3 paraphrases the requirement that the sequential strategy to begin with is fair, Conditions 1 and 2 take care of the distributed part of the derivation. Intuitively, Conditions 1 and 2 guarantee that a clause which can be generated from two persistent non-redundant clauses φ_1 and φ_2 residing at two different nodes, will be considered. Condition 2 ensures that φ_1 and φ_2 will eventually meet each other through inference messages. Condition 1 makes sure that

[2] The adjective *abstract* indicates that i is an index in the abstract view of the derivation, and not a time of any processor's clock.

all inference messages be processed ($M_\infty = \emptyset$), all raw clauses (those that, in the presence of contraction rules, remain non-trivial after having been fully contracted) become new settlers ($CP_\infty = \emptyset$) and all new settlers become residents at some place ($NS_\infty = \emptyset$). Because the definition of uniform fairness, and consequently our three conditions, focus only on persistent, non-redundant clauses, it is fairly simple to show that these three conditions imply Definition 4.1:

Theorem 4.1 *If a distributed derivation satisfies Conditions 1, 2 and 3, then it is* uniformly fair*, i.e.* $I_e(G_\infty - R(G_\infty)) \subseteq \bigcup_{k=1}^{n} \bigcup_{i \geq 0} G_i^k$.

Proof: let φ be any clause in $I_e(G_\infty - R(G_\infty))$ with parents ψ_1 and ψ_2. Since $M_\infty = CP_\infty = NS_\infty = \emptyset$, $G_\infty = S_\infty$, i.e. $\psi_1, \psi_2 \in S_\infty$. It follows that $\psi_1 \in (S_\infty^k - R(S_\infty))$ and $\psi_2 \in (S_\infty^h - R(S_\infty))$ for some $1 \leq k, h \leq p$.

If $k = h$, then $\varphi \in I_e(S_\infty^k - R(S_\infty))$. Since $S_\infty^k \subseteq S_\infty$, by the monotonicity of the redundancy criterion, $R(S_\infty^k) \subseteq R(S_\infty)$ and thus $I_e(S_\infty^k - R(S_\infty)) \subseteq I_e(S_\infty^k - R(S_\infty^k))$. By Condition 3, there exists an i such that $\varphi \in CP_i^k \subseteq G_i^k \subseteq \bigcup_{k=1}^{n} \bigcup_{i \geq 0} G_i^k$.

If $k \neq h$, let i_1 and i_2 be the abstract birth-times of ψ_1 and ψ_2 respectively. By Condition 2, we have $\psi_1 \in M_{l_1}^h$ for some $l_1 > i_2$ and $\psi_2 \in M_{l_2}^k$ for some $l_2 > i_1$. Since $M_\infty = \emptyset$ by Condition 1, we know that the inference message ψ_1 does not persist at p_h and the inference message ψ_2 does not persist at p_k. An inference message may be deleted before performing expansion steps, by a contraction step. Since ψ_1 and ψ_2 are in $G_\infty - R(G_\infty)$, i.e. they are globally persistent and non-redundant, this is impossible. It follows that the inference messages $\psi_1 \in M_{l_1}^h$ and $\psi_2 \in M_{l_2}^k$ are deleted only after having been processed. Thus, paramodulation of ψ_1 into ψ_2 is tried at p_h and paramodulation of ψ_2 into ψ_1 is tried at p_k. Either one of these two steps generates φ, i.e. either $\varphi \in CP_i^h$ or $\varphi \in CP_i^k$ at some stage i, i.e. $\varphi \in \bigcup_{k=1}^{n} \bigcup_{i \geq 0} G_i^k$. □

By this theorem, the abstract definition of uniform fairness is reduced to three more concrete requirements:

Corollary 4.1 *Let* $\mathcal{C} =< I; \Sigma >$ *be a complete (sequential) theorem proving strategy and* $\mathcal{D}$ *be its distributed version. If the algorithms and policies handling messages satisfy Conditions 1 and 2, then* $\mathcal{D}$ *is a complete distributed theorem proving strategy.*

5 Techniques to satisfy the conditions for uniform fairness

5.1 Inference messages and localized image sets

The second of the sufficient conditions for uniform fairness requires that for every persistent resident φ at node p_k, all persistent residents of other nodes appear eventually at p_k as inference messages after the abstract birth-time of φ. In the following we describe some techniques to ensure this condition. In addition to giving a general method, we also provide techniques for fine-tuning according to the parameters of different architectures.

The basic idea is that a new resident ψ, once settled, should emit a message of itself, so that inferences between residents at other nodes and ψ can be performed. We assign, to each

resident, an *identifier* and a *birth-time*. When a new settler ψ becomes a resident at p_i, it is given an identifier a never used before at p_i, and the current time at p_i's clock as its birth-time. We remark that this birth-time has no relationship with the abstract birth-time of persistent residents mentioned in previous sections; the latter is only for conceptual simplicity of the formalism. Thus, the format of a resident is $< \psi, a, x >$, where a is the identifier and x is the birth-time, and the pair $< i, a >$ represents a unique *global identifier* for ψ.

An inference message may need to be emitted when a new settler becomes a resident and when a resident is updated due to contraction. In addition to the clause, an inference message also carries its global identifier and birth-time. Thus, an inference message for $< \psi, a, x > \in S^i$ has the form $m =< \psi, i, a, x >$. If a resident ψ is contracted to ψ', its birth-time is updated: $< \psi, a, x >$ is replaced by $< \psi', a, y >$, where y is the current time at the node's clock. Intuitively, this means that ψ' is "new" and therefore should be re-scheduled to be sent as message. When node p_j receives $m =< \psi, i, a, x >$, p_j stores it in a *queue of messages*. This queue may be sorted according to different criteria, which are part of the search plan executed at the node. When the message m is selected from the queue, it is used for contraction inferences and expansion inferences (such as paramodulate *into* the residents of p_j) according to the local search plan.

One question which needs to be addressed is what p_j should do with m after p_j has used m to perform all possible inferences within its current local data base. A natural solution is to simply delete m, since the content of m already exists at node p_i and, thus, its presence (or absence) at p_j does not have any effect on the global data base. A complication arises, however, because new residents may be created at p_j *after* m has been deleted. Then, in order to ensure fairness, the content of the identifier $< i, a >$ should be known at p_j again, so that inferences between the new residents and the content of $< i, a >$ can be performed. This can be done by using "control messages" to stir the re-edition of inference messages. That is, node p_j sends a *wake-up call* to p_i requesting the content of $< i, a >$. Upon receiving the wake-up call, p_i will send the relevant information to p_j. The disadvantage of this approach is that it may generate a large amount of communication. Although by carefully analyzing the routing algorithms one can eliminate some messages [7], it may still worsen the computation/communication ratio quite significantly. On the other hand, if the architecture under consideration has very limited amount of local memory for each node and has low latency and high throughput in communication, then this scheme may be considered.

Assuming that each node has sufficient local memory, one can then choose to have each node saving the used messages in its memory. In this way, each node p_k progressively builds a *localized image set* SH^k, i.e. a local, approximate image of the global data base. Each SH^k can be implemented as a *hash table* with the global identifier of the incoming messages as the *key*: message $m =< \psi, i, a, x >$ is inserted in the hash table under key $< i, a >$. If the architecture supports an additional shared memory component, then a single *global image set* SH can be built in the shared memory, so that it can be used by all nodes, thus saving the duplication of building different hash tables. Under this scheme, there is no need for wake-up calls and replies. If a new resident is produced, it can simply go through the hash table to find the needed data to perform necessary expansion inferences. This cuts down the amount of communication significantly.

In summary, the diffusion of clauses as inference messages allows expansion steps between

clauses at remote sites. Each node p_i is responsible for sending its residents. Birth-times are used to ensure local fairness in the emission of messages. Because the data bases at the nodes are dynamic, inference messages need to be saved by the receiver or re-issued by the sender upon the receipt of wake-up calls.

5.2 Deletion of redundant inference messages

The above techniques, however, do introduce some redundancy. The reason is that during a derivation it is not known which residents are persistent. Thus, inference messages may carry residents for which the messages are not necessary, because these residents are contracted after the generation of the message. For instance, assume that ψ is a resident at p_i and that a message $m_1 =< \psi, i, a, x >$ has been emitted for ψ. Suppose later p_i has additional data enabling it to contract ψ to ψ'. Then, another message $m_2 =< \psi', i, a, y >$ for ψ' is sent. A node p_j, which receives both m_1 and m_2, does not know that m_1 is redundant and may use both messages for inferences, thus producing redundancy. Furthermore, a contraction-based distributed strategy may prescribe to forward inference messages only after having tried to contract their content. If message m_1 from p_i reaches p_j by going through intermediate nodes, node p_j will receive in general a message $< \psi'', i, a, x >$, where ψ'' is a contracted form of ψ. Clause ψ'' may actually render ψ' redundant. One needs to design a way to take full advantage of the results of these contractions.

A solution to these problems lies in the utilization of the global identifier and birth-time which come with the messages. Suppose node p_j has received two messages $m_1 =< \psi_1, i, a, x >$ and $m_2 =< \psi_2, i, a, y >$, with the same global identifier $< i, a >$. Then, we know that ψ_1 and ψ_2 are *logically equivalent* since they are (contracted) forms of the same resident. We term such messages *generalized duplicates*. If $\psi_1 = \psi_2$ and $x = y$, the two messages are just two plain duplicates. If $\psi_1 \neq \psi_2$, but $x = y$, then the two messages were originally plain duplicates, whose clauses have been contracted to two different forms during their traversal through the network. If $x < y$, then the original clause of the message m_2 is a contracted form of the original clause of m_1, since m_2 is emitted later. The case for $x > y$ is symmetric.

Let $m_1 =< \psi_1, i, a, x >$ and $m_2 =< \psi_2, i, a, y >$ be two generalized duplicates such that $y \geq x$. If they carry the same clause, i.e. $\psi_1 = \psi_2$, then we discard m_1, the message carrying an earlier time-stamp. Otherwise, we discard m_1 if $\psi_1 \succ \psi_2$ and discard m_2 if $\psi_2 \succ \psi_1$. If ψ_1 and ψ_2 are not comparable, then we discard m_1 since it was emitted earlier. The following inference rule captures the above ideas:

Discard Message: Let i and k be two nodes, and $y \geq x$.

- $\dfrac{M^k \cup \{< \psi_1, i, a, x >, < \psi_2, i, a, y >\}}{M^k \cup \{< \psi_2, i, a, y >\}} \quad if\ \psi_2 \not\succ \psi_1$
- $\dfrac{M^k \cup \{< \psi_1, i, a, x >, < \psi_2, i, a, y >\}}{M^k \cup \{< \psi_1, i, a, x >\}} \quad if\ \psi_2 \succ \psi_1$

These inference rules formalize a mechanism which discards redundant inference messages at

the receiver. By the nature of the theorem proving applications, it is not possible to discard generalized duplicates at the sender. In fact, it is never the case that two generalized duplicates $< \psi_1, i, a, x >$ and $< \psi_2, i, a, y >$ be present at the same time at the sender, i.e. node p_i, because the content of $< i, a >$ is only one clause at any given time. The issue may be addressed at the sender in terms of the delay between contraction and communication. On one hand, when a resident is contracted, we would like to send it as inference message as soon as possible to let the other processes see the reduced form. On the other hand, we may prefer to wait for a longer interval, in case the resident will be reduced again shortly. A suitable trade-off may be determined empirically. The infererence rule of Discard Message may be applied to other aspects of the Clause diffusion methodology, such as for updating the contents of the image sets [7].

6 Discussion

In this paper we outlined a general approach to distribued automated deduction terms "clause-diffusion", and we described the related problem of fairness in a distributed derivation, and how to ensure it in our methodology. Although our work applies to distributed theorem proving methods in general, we have concentrated on contraction-based strategies, since they pose the most challenging problems both in theory and in implementation. The Clause-Diffusion methodology is a general method which can be implemented on a variety of architectures and yields a high degree of parallelism by tolerating extra global redundancy, as the same clauses may be generated in different ways by different processes. Our view is that it is better to let the processes proceed eagerly in parallel, generating extra redundant clauses and deleting them afterwards, rather than synchronize the processes, thus forcing them to wait, in order to prevent redundancies. Next, we applied the notion of uniform fairness from [4] to the distributed framework. We pointed out the new problem on fairness represented by the additional component of communication and presented a set of sufficient conditions to ensure fairness. Our conditions are fairly general and the correctness is proved.

We then discussed how these sufficient conditions can be realized on various architectures. The problem of redundancy resulting from the inference messages is also discussed in detail and technical solutions are described.

Not much work has been done in distributed theorem proving. The DARES system [9] and the team-work method [10] apply to theorem proving artificial intelligence techniques for distributed problem solving. Theoretical treatments were not given in the papers. DARES has been designed for strategies without backward contraction. The team-work method relies on a central control to synchronize periodically the processes and evaluate their work, whereas our approach is intrinsically distributed and asynchronous. Therefore, most of the results presented here appear to be new. They include the Clause-Diffusion methodology for distributed deduction, the sufficient conditions for ensuring fairness and the techniques to implement them.

We also feel that part of our study may be applicable to distributed data bases. One of the important problems in distributed data bases is to maintain the global consistency of data in the presence of updates. The same problem also appears in our study although in a weaker

form. Contraction inferences are in fact a form of update, since they replace data by others. Our counterpart to the notion of consistency, on the other hand, is not as rigid. The main difference is that once a datum is modified through contraction, it is not necessary to require that all of its copies (in the form of inference messages) be updated immediately into identical form. This is because a contracted datum is still *logically equivalent* to the original one, which is all that is required in automated deduction. What is lost by not keeping copies of the same datum identical is not consistency, but minimality: there is a possible temporary increase of redundancy which, although undesirable, does not disturb the global integrity of the system. This is why it is relatively easier to come up with a reasonable solution for our problem than for the similar problem for distributed data bases. Some of our work may be useful to distributed data bases applications with less stringent requirements of consistency.

References

[1] S.Anantharaman, J.Hsiang, Automated Proofs of the Moufang Identities in Alternative Rings, *JAR*, Vol. 6, No. 1, 76–109, 1990.

[2] L.Bachmair, N.Dershowitz and J.Hsiang, Orderings for Equational Proofs, in *Proc. of LICS-86*, 346–357, 1986.

[3] L.Bachmair, Proofs Methods for Equational Theories, Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois at Urbana, 1987.

[4] L.Bachmair, H.Ganzinger, Non-Clausal Resolution and Superposition with Selection and Redundancy Criteria, in *Proc. of LPAR-92*, LNAI 624, 273–284, 1992.

[5] M.P.Bonacina, J.Hsiang, On fairness of completion-based theorem proving strategies, in R.V. Book (ed.), *Proc. of RTA-91*, LNCS 488, 348–360, 1991.

[6] M.P.Bonacina, J.Hsiang, Towards a Foundation of Completion Procedures as Semidecision Procedures, submitted and Tech. Rep. Dept. of Computer Science, SUNY at Stony Brook, Aug. 1991.

[7] M.P.Bonacina, Distributed Automated Deduction, Ph.D. Thesis, Dept. of Computer Science, SUNY at Stony Brook, Dec. 1992.

[8] J.D.Christian, High-Performance Permutative Completion, Ph.D. Thesis, Univ. of Texas at Austin, and MCC Tech. Rep. ACT-AI-303-89, Aug. 1989.

[9] S.E.Conry, D.J.MacIntosh and R.A.Meyer, DARES: A Distributed Automated REasoning System, in *Proc. of AAAI-90*, 78–85, 1990.

[10] J.Denzinger, Distributed knowledge-based deduction using the team work method, Tech. Rep., Univ. of Kaiserslautern, 1991.

[11] D.J.Hawley, A Buchberger Algorithm for Distributed Memory Multi-Processors, in *Proc. of the Int. Conf. of the Austrian Center for Parallel Computation*, Linz, Oct. 1991.

[12] J.Hsiang, M.Rusinowitch, On word problems in equational theories, in Th.Ottman (ed.), *Proc. of ICALP-87*, LNCS 267, 54–71, 1987.

[13] G.Huet, A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm, *JCSS*, Vol. 23, 11–21, 1981.

[14] A.Jindal, R.Overbeek and W.Kabat, Exploitation of parallel processing for implementing high-performance deduction systems, *JAR*, Vol. 8, 23–38, 1992.

[15] D.Kapur. H.Zhang, RRL: a Rewrite Rule Laboratory, in E.Lusk, R.Overbeek (eds.), *Proc. of CADE-9*, LNCS 310, 768–770, 1988.

[16] C.Kirchner. P.Viry, Implementing Parallel Rewriting, in B.Fronhöfer and G.Wrightson (eds.), *Parallelization in Inference Systems*, LNAI 590, 123–138, 1992.

[17] D.W.Loveland, *Automated Theorem Proving: A Logical Basis*, North-Holland, Amsterdam, 1978.

[18] E.L.Lusk, W.W.McCune, Experiments with ROO: a Parallel Automated Deduction System, in B.Fronhöfer and G.Wrightson (eds.), *Parallelization in Inference Systems*, LNAI 590, 139–162, 1992.

[19] W.W.McCune, OTTER 2.0 Users Guide, Tech. Rep. ANL-90/9, Argonne National Lab., Mar. 1990.

[20] M.Rusinowitch, Theorem-proving with Resolution and Superposition, *JSC*, Vol. 11, No. 1 & 2, 21–50, Jan./Feb. 1991.

[21] K.Siegl, Gröbner Bases Computation in STRAND: A Case Study for Concurrent Symbolic Computation in Logic Programming Languages, M.S. Thesis and Tech. Rep. 90-54.0, RISC-LINZ, Nov. 1990.

[22] R.Socher-Ambrosius, How to Avoid the Derivation of Redundant Clauses in Reasoning Systems, in *JAR*, Vol. 9, No. 1, Aug. 1992.

[23] M.E.Stickel, The Path-Indexing Method for Indexing Terms, Tech. Note 473, SRI Int., Oct. 1989.

[24] J.-P.Vidal, The Computation of Gröbner Bases on A Shared Memory Multiprocessor, in A.Miola (ed.), *Proc. of DISCO-90*, LNCS 429, 81–90, 1990 and Tech. Rep. CMU-CS-90-163, Aug. 1990.

[25] K.A.Yelick, S.J.Garland, A Parallel Completion Procedure for Term Rewriting Systems, in D.Kapur (ed.), *Proc. of CADE-11*, LNAI 607, 109–123, 1992.