J. Symbolic Computation (1996) **21**, 507–522

On the Reconstruction of Proofs in Distributed Theorem Proving: a Modified Clause-Diffusion Method

MARIA PAOLA BONACINA [†]

Department of Computer Science

University of Iowa

Iowa City, IA 52242-1419, U.S.A.

bonacina@cs.uiowa.edu

(Received 31 May 1995)

Proof reconstruction is the operation of extracting the computed proof from the trace of a theorem-proving run. We study the problem of proof reconstruction in *distributed theorem proving*: because of the distributed nature of the derivation and especially because of deletions of clauses by *contraction*, it may happen that a deductive process generates the empty clause, but does not have all the necessary information to reconstruct the proof. We analyze this problem and we present a method for distributed theorem proving, called *Modified Clause-Diffusion*, which guarantees that the deductive process that generates the empty clause will be able to reconstruct the distributed proof. This result is obtained without imposing a centralized control on the deductive processes or resorting to a round of post-processing with ad hoc communication. We prove that Modified Clause-Diffusion is fair (hence complete) and guarantees proof reconstruction. First we define a set of conditions, next we prove that they are sufficient for proof reconstruction, then we show that Modified Clause-Diffusion satisfies them. Fairness is proved in the same way, which has the advantage that the sufficient conditions provide a treatment of the problem relevant for distributed theorem proving in general.

1. Introduction

Proof reconstruction is an important feature of theorem provers that implement resolution-based or completion-based strategies. These strategies work primarily by forward reasoning, that is, by deriving consequences from the axioms and the negation of the target theorem, until a contradiction, the empty clause, is generated. Contraction rules such as simplification and subsumption are employed to delete those generated clauses that are redundant. While searching for the empty clause, these procedures typically produce a very high number of clauses, many of whom may not contribute directly to deriving the empty clause. The numbers of clauses vary with the theorem-proving problem and the strategy, but theorem-proving runs that involve millions of clauses

[†] Supported in part by the National Science Foundation with grant CCR-94-08667.

are not regarded as exceptional. Since an output of this size is unpractical for most purposes, theorem provers incorporate an algorithm that extracts from the record of all the generated clauses those that are related by inference steps to the empty clause. This process is called *proof reconstruction*.

The motivation for proof reconstruction is that theorem provers are expected to produce an output that their users may understand. A yes/no answer is not sufficient in most contexts. The entire trace of the derivation is often too long to be readable. Therefore, theorem provers need to include proof-presentation features that make the computed proof accessible. The capability of extracting the proof from the derivation is obviously a prerequisite for proof presentation.

In this paper, we study proof reconstruction in *distributed theorem proving*. By distributed theorem proving, we mean having multiple concurrent, asynchronous deductive processes working in parallel on the same theorem-proving problem. Each process executes a theorem-proving strategy, has its own database of clauses and develops its own derivation. The processes may all execute the same strategy or execute strategies with different search plans, e.g. different criteria to select inference rules and premises. A method for distributed theorem proving specifies, together with the strategy or strategies to be executed by the processes, a mechanism to subdivide the theorem-proving problem among the processes, and a communication scheme: the former aims at ensuring that each process has less work than a single sequential process would; the latter aims at ensuring that the processes cooperate, for instance by exchanging the clauses they derive. In this context, a distributed derivation is made by the collection of the derivations developed by the processes and it succeeds when one of the processes generates the empty clause. We refer to Bonacina and Hsiang (1994) and Suttner and Schumann (1994) for surveys on parallel and distributed deduction for different strategies and architectures, and to Bonacina and Hsiang (1995a) for the Clause-Diffusion method, which we adopt here as a starting point.

In sequential theorem proving, proper book-keeping is sufficient to guarantee proof reconstruction. The situation is sensibly different in distributed theorem proving. The distributed nature of the derivation implies that while one process succeeds first, all processes contributed to the proof, and it is not trivial to guarantee that the successful process is capable of reconstructing the proof by consulting only its own database. As an example, consider the following scenario: a deductive process p_i generates an equation φ and applies it to reduce another clause ψ to a new form ψ'. It follows that φ and ψ are parents of ψ'. Then, process p_i also simplifies φ itself to φ'. Later, φ' and ψ' are sent by process p_i to another process p_j. Eventually, p_j generates the empty clause, and the proof involves ψ' at some stage. When p_j tries to reconstruct the proof, the history of ψ' will refer to φ and ψ, but neither of them can be retrieved in the database of p_j.

In this work we give a systematic treatment of the problem of proof reconstruction in distributed theorem proving, and we propose a solution that prevents situations such as the one illustrated by the above example. First, we define formally proof reconstruction and we overview briefly the Clause-Diffusion approach. Then we classify the possible failures in distributed proof reconstruction. These observations guide the design of the Modified Clause-Diffusion method, so that proof reconstruction failures are avoided. In the formal sections of the paper, we prove that modified Clause-Diffusion is fair, and therefore complete, if the underlying inference system is complete, and that it guarantees proof reconstruction. We demonstrate the latter property by formulating sufficient conditions for distributed proof reconstruction: these conditions apply to general distributed theorem proving, beyond Clause-Diffusion itself. We show that our conditions are sufficient and that the Modified Clause-Diffusion method fulfils them.

We remark that Modified Clause-Diffusion guarantees proof reconstruction by using the asynchronous communication that is in place for the distribution of the work-load. No ad hoc communication for proof reconstruction is needed. Also, Modified Clause-Diffusion preserves the characteristic of Clause-Diffusion that all deductive processes are asynchronous peers. No central control, such as in a master-slave type of organization, where the master performs centralized book-keeping and decision-making, is added. The capability of proof reconstruction is ensured by schemes for communication, identification, and allocation of clauses, that are executed in a purely distributed, asynchronous fashion by the processes.

To our knowledge, the problem of proof reconstruction in distributed theorem proving with loosely coupled, asynchronous peer processes and separate databases was not considered before. In general, the more centralized the control is and the more predictable the communication is, the simpler is the book-keeping for proof reconstruction. Thus, proof reconstruction in distributed memory is more difficult than proof reconstruction in parallel theorem proving in shared memory, because in the latter there is only one database in the shared memory and proof reconstruction can be done as in the sequential case. Similarly, proof reconstruction in a distributed system with peer processes is different than proof reconstruction in a distributed system with a hierarchical organization: if the processes work as master and slaves, it is sufficient to reconstruct the proof in the database of the master. In the Team-Work method of Avenhaus and Denzinger (1993), the databases of the deductive processes are periodically merged, so that proof reconstruction can also be done in a single database (Denzinger and Schulz 1994). The work reported in this paper appeared in preliminary form in Bonacina (1994).

2. Proof Reconstruction

This section contains the definitions of computed proof, proof reconstruction and conditions for proof reconstruction in sequential theorem proving. We assume to have a theorem-proving strategy $\mathcal{C} = \langle I; \Sigma \rangle$, where I is the set of inference rules and Σ is the search plan that controls the application of the inference rules. The inference rules comprise both *expansion rules*, such as resolution and paramodulation, and *contraction rules*, such as equational simplification and subsumption. Given a theorem-proving problem $S \models \varphi_0$ in refutational form ($S_0 = S \cup \{\neg\varphi_0\}$), the strategy will generate a derivation

$$S_0 \underset{\mathcal{C}}{\vdash} S_1 \underset{\mathcal{C}}{\vdash} \dots S_i \underset{\mathcal{C}}{\vdash} S_{i+1} \underset{\mathcal{C}}{\vdash} \dots,$$

where at each step an inference rule is applied to selected premises according to the search plan. For each generated clause φ, the "proof" or "justification" of φ is made of the inference steps that derived φ from the input clauses. We represent it as an *ancestor-tree*:

Definition 2.1. *Let $\mathcal{D}$ be a derivation $S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \dots \vdash_{\mathcal{C}} S_i$ .... For all clauses $\varphi \in \bigcup_{i \geq 0} S_i$, the* ancestor-tree *of φ in $\mathcal{D}$, denoted by $at_{\mathcal{D}}(\varphi)$, is a tree with root labelled φ and no subtrees, if $\varphi \in S_0$, with subtrees $at_{\mathcal{D}}(\varphi_1), \dots, at_{\mathcal{D}}(\varphi_n)$ if φ is generated at stage $i > 0$ from premises $\varphi_1, \dots, \varphi_n$.*

For instance, if φ is a resolvent of φ_1 and φ_2, $at_{\mathcal{D}}(\varphi)$ has root φ and subtrees $at_{\mathcal{D}}(\varphi_1)$ and $at_{\mathcal{D}}(\varphi_2)$. This representation applies also to contraction inferences that generate clauses, such as equational simplification. For example, if φ is generated as the normal form of a pre-existing clause ψ with respect to the equations $\varphi_1, \ldots, \varphi_n$, $at_{\mathcal{D}}(\varphi)$ has root φ and subtrees $at_{\mathcal{D}}(\varphi_1), \ldots, at_{\mathcal{D}}(\varphi_n)$ and $at_{\mathcal{D}}(\psi)$. Thus, the ancestor-tree of a clause contains all its ancestors, including both expansion-ancestors, i.e. clauses used as parents in expansion steps, and contraction-ancestors, i.e. simplifiers and ancestors that were reduced. To complete the representation, node φ in $at_{\mathcal{D}}(\varphi)$ may also be decorated by a label denoting the applied inference rule.

We remark that *variants*, i.e. clauses that differ only by a renaming of variables, are regarded as distinct clauses. This assumption is reasonable, because in practice theorem provers do treat variants as distinct clauses. Under this assumption, the same clause is never derived twice, since each clause has its own set of variables. If a clause φ is derived at stage i and a variant φ' of φ is derived at stage j, their ancestor-trees are two distinct objects, even if they may represent the same inferences logically. It follows that the ancestor-tree of a clause in a given derivation is unique. On the other hand, an ancestor-tree may have more than one node labelled by the same clause, since a premise may be used more than once to generate a clause.

The proof computed by a derivation can then be defined as the ancestor-tree of the empty clause:

Definition 2.2. *If $\mathcal{D}$ is a successful derivation $S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \ldots S_i \vdash_{\mathcal{C}} S_{i+1} \ \ldots \vdash_{\mathcal{C}} S_h$, the* proof computed by the derivation *is $at_{\mathcal{D}}(\Box)$.*

We shall assume that the theorem prover builds ancestor-trees correctly, and the information about the used inference rules is stored with the clauses, so that proof reconstruction reduces to retrieval of ancestors. In order to reconstruct the computed proof, and also for other reasons, including selection by the search plan, theorem provers associate *identifiers* to clauses. Identifiers are chosen from a countably infinite ordered set, since infinitely many clauses may be generated. Often $\mathbb{N}$ itself (the natural numbers) is used. We call *naming scheme* the mechanism that a theorem prover uses to associate identifiers to clauses:

Definition 2.3. *Let $\mathcal{L}$ be the language of clauses on the given signature. A* naming scheme *is a pair (A, R), where A is a countably infinite ordered set, and R is a relation $R \subseteq A \times \mathcal{L}$, called* retrieval relation*, such that $(x, \varphi) \in R$ means that x is the identifier of clause φ.*

The relevant property for proof reconstruction is the following:

Definition 2.4. *A theorem-proving strategy $\mathcal{C}$ has an* unambiguous *naming scheme (A, R) if, for all derivations by $\mathcal{C}$, $S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \ldots S_i \vdash_{\mathcal{C}} S_{i+1} \ \ldots$, $R\colon A \rightarrow \bigcup_{i \geq 0} S_i$ is a bijective function.*

R needs to be a function, so that, given an identifier x, there exists one and only one clause $\varphi \in \bigcup_{i \geq 0} S_i$ that is identified by x. Bijectivity means that every generated clause has an identifier and such identifier is unique. Multiple variants of a clause are treated as distinct clauses and are given different identifiers.

An unambiguous naming scheme is essentially sufficient for proof reconstruction in sequential theorem-proving. For instance, the theorem prover Otter (McCune 1994) uses an unambiguous naming scheme and a data structure for representing clauses with fields to store the identifier of the clause, the identifiers of its parent clauses and the code of the inference rule that generated it. When an empty clause $\Box$ is produced, the prover reconstructs $at_{\mathcal{D}}(\Box)$ by retrieving the parents of $\Box$, then the parents of the parents and so on, until the reconstruction process reaches clauses that were part of the input set.

Additional care is needed if the strategy features contraction inference rules, because clauses deleted by contraction may occur in $at_{\mathcal{D}}(\Box)$ and thus may be needed for the purpose of proof reconstruction even if they are no longer used for inferences. We distinguish between *forward contraction*, the contraction of newly generated clauses right after generation, and *backward contraction*, the contraction of all other clauses. Clauses deleted by forward contraction are not used as premises of other steps before deletion and therefore cannot occur in $at_{\mathcal{D}}(\Box)$. Clauses deleted by backward contraction may occur in $at_{\mathcal{D}}(\Box)$, because they may have been used as premises of other steps before being deleted. Therefore, the clauses deleted by backward contraction need to be saved in a separate component D of the database, which will be consulted only by the proof reconstruction algorithm. The derivation assumes the form:

$$(S_0; D_0) \underset{\mathcal{C}}{\vdash} (S_1; D_1) \underset{\mathcal{C}}{\vdash} \dots (S_i; D_i) \underset{\mathcal{C}}{\vdash} \dots.$$

Theorem provers such as Otter proceed in this way for the sake of proof reconstruction, with no apparent harm for performance.

3. Distributed Theorem Proving

In this section, we describe the type of distributed theorem proving, theorem proving by Clause-Diffusion, that is the context for our study of proof reconstruction.

Clause-Diffusion seeks to realize a form of coarse-grain parallelism for theorem proving called *parallelism at the search level* (Bonacina and Hsiang 1994). The idea is to have concurrent deductive processes $p_0, \dots, p_{n-1}$ searching in parallel the search space of the theorem-proving problem. We assume a distributed environment, with distributed memory and message-passing, where each process runs on a node of the system, also denoted by $p_0, \dots, p_{n-1}$. Given a theorem-proving strategy $\mathcal{C}$ and an input problem S_0, each process p_k executes the strategy, generating its derivation

$$S_0^k \underset{\mathcal{C}}{\vdash} S_1^k \underset{\mathcal{C}}{\vdash} \dots S_i^k \dots.$$

The *distributed derivation* is formed by the collection of these derivations and it succeeds as soon as one of them does. The set S_i^k represents the *local database* of process p_k at stage i, and $S_i = \bigcup_{k=0}^{n-1} S_i^k$ represents the *global database* at stage i. The partition of S_i in the S_i^k's is the *physical partition* of the database, because the clauses in S_i^k are physically stored at p_k. The physical partition is not a partition in the mathematical sense, since the S_i^k's generally have non-empty intersections.

For the purpose of subdividing the search space among the processes, each clause is assigned to a process and is said to be a *resident* of that node. This partition is called *logical partition*, and it is a true partition (under the assumption that variants are distinct clauses), because each clause belongs only to one process. Then, each process performs only those inferences that involve its residents. For instance, for the paramodulation inference rule, process p_k performs only those paramodulation steps that paramodulate into a resident of p_k. For two clauses ψ_1 and ψ_2, belonging to p_k and p_h respectively, p_k will paramodulate ψ_2 into ψ_1, whereas p_h will paramodulate ψ_1 into ψ_2. Similar criteria for subdivision apply to resolution, hyperresolution, and other expansion inference rules. The clauses thus generated are called *raw clauses*. Every raw clause is forward-contracted and assigned to a process by executing the *allocation algorithm* that controls the logical partition. Inferences between clauses belonging to different processes are made possible by broadcasting, or *diffusing* the clauses (hence the name of the methodology) in the form of *inference messages*. It follows that the database of each process contains both residents and non-resident clauses that were received as inference messages.

While it applies to theorem proving in general, the Clause-Diffusion methodology targets primarly *contraction-based* strategies, that is, strategies with contraction rules and an *eager-contraction* search plan. For these strategies it is fundamental to address the issue of *distributed global contraction*, the contraction of clauses (both forward and backward) with respect to the distributed global database. One approach is to let the processes save the clauses received as inference messages, and form with them an approximated version, termed *localized image set*, of the current state of the global database. Each process uses its localized image set as set of simplifiers for the purpose of distributed global contraction. Another key aspect of contraction-based strategies is that a reduced clause generated by backward contraction is also regarded as a raw clause, that needs to be tested for further contraction with respect to the distributed global database.

A specific Clause-Diffusion strategy is defined by specifying its components, including the inference system, the search plan, the algorithm for the allocation of clauses to processes, the scheme for communication of the inference messages and the scheme for distributed global contraction. The interested reader may find in Bonacina and Hsiang (1995a) a complete presentation of Clause-Diffusion, and in Bonacina and Hsiang (1995b) and in Bonacina and McCune (1994) specific strategies. In this paper we do not assume a specific strategy, because we wish our study of proof reconstruction to be as general as possible.

4. The Problems in Distributed Proof Reconstruction

In this section we identify the types of problems that may cause failures in reconstruction of proofs. We find that proof reconstruction depends mainly on three components of a distributed strategy: the naming scheme, the communication scheme and the treatment of clauses generated by backward contraction.

4.1. failures by name clash

The basic issue is the same as in sequential theorem proving: the naming scheme should be unambiguous. If different clauses φ and ψ receive the same identifier x, and a reference to x is found in the process of reconstructing the proof, proof reconstruction will fail, because it cannot be resolved whether the occurrence of x refers to φ or ψ. We call such failure a *failure by name clash*. In distributed theorem proving, the naming scheme needs to be unambiguous not only within the local database of each deductive process, but also in the global database. Since multiple processes name clauses concurrently, care must be taken that different clauses do not receive the same name. This involves several issues including: how to subdivide the task of naming clauses among the processes (e.g., should a process name the clauses it owns? or the clauses it generates? or the clauses it holds in its local database?); how the naming scheme interplays with communication (e.g., may a process change the name of a clause it received as an inference message?); how the naming scheme interplays with backward contraction (e.g., what happens to the name of a clause when the clause is backward-contracted?). For instance, consider a clause φ with identifier x, resident at a node p_i, that is broadcast at some stage of the derivation. At some later stage another process p_k simplifies φ to φ'. Assume that p_k discards φ and keeps x as the name of φ'. This may be because p_k does not own φ and φ', or because p_k broadcasts φ' without keeping it. Clause φ' will be used as premise, so that the ancestor-trees of other clauses may contain occurrences of x referring to φ'. If p_k, or some other process p_j that receives φ' from p_k, finds a proof including a reference to φ, proof reconstruction fails if the identifier x retrieves φ' instead of φ. Symmetrically, if p_i, or some other process unaware of φ', finds a proof containing a reference to φ', proof reconstruction fails if the identifier x retrieves φ instead of φ'.

4.2. FAILURES BY DELAYED DIFFUSION

The second type of failure is related to the communication scheme. Proof reconstruction may fail if a clause is sent or broadcast earlier than one of its ancestors. For instance, consider a clause φ in the database of p_k. Process p_k generates another clause ψ from φ by either expansion or contraction, so that φ is a parent of ψ. Assume that p_k broadcasts ψ *before* φ, or that p_k broadcasts ψ, but not φ, because φ is deleted by backward contraction. It follows that some other process p_h may receive ψ and find a proof involving ψ before receiving φ. Process p_h will not be able to reconstruct the proof, because the reference to the identifier of φ in ψ cannot be solved. We call this phenomenon *failure by delayed diffusion*, because it may happen if clauses are diffused too late. In experiments, we also observed that a communication scheme with both "send" and "broadcast" operations may cause failures by delayed diffusion if "send" is much faster than "broadcast".

5. Modified Clause-Diffusion

In this section we describe the features of Modified Clause-Diffusion that prevents the proof reconstruction failures of the previous section.

5.1. THE NAMING SCHEME

In Modified Clause-Diffusion, a clause is given its identifier by the process that generates it. Whenever process p_k generates a raw clause φ, p_k reduces φ to its normal form φ' (forward contaction), and if φ' is not deleted, p_k executes the allocation algorithm to decide which process φ' belongs to. Let p_j be this process. Then the identifier of φ' is $\langle j, k, l\rangle$ if φ' is the l-th clause to be allocated as resident to p_j among all those generated by p_k. As a special case, if p_k allocates the clause to itself, the identifier will have the form $\langle k, k, l\rangle$, with the same meaning for l. This naming scheme is *unambiguous*, because no two processes may generate the same identifier and no process may generate the same identifier twice. A naming scheme that uses only two components, where the identifier of φ' is $\langle k, l\rangle$, if φ' is the l-th clause generated by p_k, is also unambiguous, but we use $\langle j, k, l\rangle$, because processes need to know which clauses they own.

5.2. THE COMMUNICATION SCHEME

Modified Clause-Diffusion prevents failures by delayed diffusion by adopting an *eager* communication scheme, where clauses are broadcast right after forward contraction. Continuing with the above description, if the allocation algorithm assigns φ' to p_k, then p_k keeps it as its resident and also broadcasts it as an inference message to the other processes. If φ' is assigned to another node p_j, then p_k keeps and broadcasts φ', realizing in one operation the goal of sending φ' to its owner p_j and the goal of broadcasting φ' to all the processes. Failures by delayed diffusion do not occur, because clauses are broadcast before being used as premises.

5.3. THE TREATMENT OF CLAUSES GENERATED BY BACKWARD CONTRACTION

An unambiguous naming scheme prevents failures by name clash only if every raw clause gets a new identifier generated by the naming scheme. For the raw clauses generated by backward contraction, this requires some additional thought. Assume that a clause φ, that was broadcast at some stage of the derivation and is stored at all the nodes, becomes reducible to a new normal form φ'. If all the processes are allowed to reduce φ to φ', up to n copies of φ' will be generated, forward-contracted, given a (different) identifier and broadcast. This clearly represents a high degree of redundancy. If, on the other hand, we establish that a process may simplify only its residents, only the owner of φ will reduce it and name and broadcast φ'. Upon receiving the inference message φ', the other processes will use it to replace φ in their databases (φ' carries in its history the information that it was generated by backward contraction of φ.). The disadvantage of this second scheme is that backward contraction is delayed, contrary to the eager-contraction search plan. Since a large part of the database of a process may be made eventually of non-resident clauses, the limitation of the contraction power of the processes is significant. Also, it complicates the treatment of inference messages, because a process needs to recognize that an incoming inference message carries the reduced form φ' of a clause φ in its database.

Modified Clause-Diffusion proposes a compromise between these two options. Each process may perform backward contraction of its own clauses by any contraction rule. It may apply without restrictions those contraction rules, such as subsumption and tautology deletion, that do not produce new clauses. In addition, it may use simplification to delete clauses belonging to other processes, but it is not allowed to generate their reduced forms. Thus, all processes may apply backward contraction to detect that φ is reducible and delete it, but only the owner of φ is allowed to complete the backward contraction inference, generate φ', name it and broadcast it. At all the other nodes the contraction step initiated by deleting φ will be completed when the inference message φ' is received. Deleting φ without generating φ' is incomplete locally, but it is complete globally, as long as φ' is generated by the owner of φ and broadcast[†].

This approach has several advantages. First, it does not induce the duplication of unrestricted backward contraction, without strongly reducing the contraction power of the

[†] This scheme can easily accomodate a strategy that separates the goal from the other clauses, such as Unfailing Knuth–Bendix Completion (Hsiang and Rusinowitch 1987, Bachmair *et al.* 1989) applied to a purely equational problem. The target theorem $\forall x s \simeq t$ is negated and skolemized into a ground inequality $\hat{s} \neq \hat{t}$ and is proved by reducing $\hat{s}$ and $\hat{t}$ to the same form. Since the goal is used for no other purpose, it can be given a special status, and all processes can keep a copy of it and reduce it.

processes, since they can still delete redundant clauses regardless of ownership. Second, there is no distinction between generation of raw clauses by backward contraction and generation of raw clauses by expansion. All generation of raw clauses is restricted based on ownership. All raw clauses are processed in the same way, and all inference messages are treated in the same way. Finally, this scheme implies that all clauses generated by backward contraction are assigned new identifiers. Together with the fact that the naming scheme is unambiguous, this means that there are no failures of proof reconstruction by name clash.

5.4. DISTRIBUTED DERIVATIONS

We summarize the operations of Modified Clause-Diffusion in a refined description of its derivations. A distributed derivation is made of a collection of n derivations

$$T_0^k \underset{\mathcal{C}}{\vdash} T_1^k \underset{\mathcal{C}}{\vdash} \dots T_i^k \underset{\mathcal{C}}{\vdash} \dots,$$

for $0 \leq k \leq n-1$, by the processes $p_0, p_1, \dots, p_{n-1}$. Here and in the rest of the paper T^k is the tuple $(S^k; V^k; CP^k; MI^k; MO^k; D^k)$ where S^k is the set of residents of p_k, V^k is the set of non-resident clauses currently held at p_k, CP^k is the set of raw clauses, MI^k is the set of inference messages being received (input), MO^k is the set of inference messages to be broadcast (output) and D^k is the set of clauses deleted by backward contraction. The different types of operations work as follows for each p_k.

1 *Expansion* takes premises in $S^k \cup V^k$ and puts the generated raw clauses in CP^k. Expansion inferences are subdivided according to the logical partition (see Section 3). For instance, for paramodulation, for any two clauses $\psi_1, \psi_2 \in S^k$, p_k paramodulates ψ_1 into ψ_2 and ψ_2 into ψ_1. For any two clauses $\psi_1 \in S^k$ and $\psi_2 \in V^k$, p_k paramodulates ψ_2 into ψ_1. (If p_h is the process that owns ψ_2 – hence $\psi_2 \in S^h$ – paramodulation of ψ_1 into ψ_2 is done by p_h when $\psi_1 \in V^h$.)
2 *Forward contraction* applies the clauses in $S^k \cup V^k$ to contract the raw clauses in CP^k. Deleted clauses are discarded, whereas for a non-trivial normal form p_k executes the allocation algorithm: if the clause is assigned to p_k, it is stored in S^k, otherwise in V^k. In either case it is also put in MO^k as an inference message to be broadcast.
3 *Backward contraction* keeps $S^k \cup V^k$ inter-reduced, by contracting clauses in $S^k \cup V^k$ with respect to $S^k \cup V^k$ itself. The generated raw clauses are treated like in the previous case, except that deleted clauses are moved to D^k.
4 The act of *broadcasting* an inference message is initiated by putting the clause in MO^k; the effect of broadcasting is represented in the derivation by the clause appearing in the MI components of all the other processes at the next stage.
5 Process p_k *receives* an inference message by moving it from MI^k to S^k, if the clause belongs to p_k, to V^k otherwise.

All clauses in the V component are copies, or “images”, of clauses in the S component:

LEMMA 5.1. *For all k, $0 \leq k \leq n-1$, for all $i \geq 0$, if $\varphi \in V_i^k$, then $\varphi \in S_j^h$, for some h, $0 \leq h \neq k \leq n-1$, and $j \geq 0$.*

PROOF. It follows trivially from Items 2, 4 and 5 above. □

Similarly, all clauses in MI^k and MO^k are copies, and clauses are in CP^k only temporarily. Thus, from a logical point of view, $\bigcup_{k=0}^{n-1} S^k$ is the global database. The union $S^k \cup V^k$ forms the localized image set of process p_k, that is, the "image" of $\bigcup_{k=0}^{n-1} S^k$ known to p_k.

5.5. uniform fairness of modified clause-diffusion

A proof of fairness of the original Clause-Diffusion method can be found in Bonacina and Hsiang (1995a). Since the method and the formal description of the derivations are different, we need to prove separately the fairness of Modified Clause-Diffusion. Fairness of a theorem-proving strategy means that the inferences that are necessary to prove the theorem will not be postponed indefinitely by the search plan of the strategy. A stronger property, that we call *uniform fairness*, says that all expansion inferences from persistent, non-redundant premises will be considered eventually by the search plan:

Definition 5.1. *(Bachmair and Ganzinger 1992) Given a set of expansion inference rules I_e and a redundancy criterion R, a derivation $S_0 \vdash S_1 \vdash \dots S_i \vdash S_{i+1} \dots$ is* uniformly fair *with respect to I_e and R if $I_e(S_\infty - R(S_\infty)) \subseteq \bigcup_{i\geq 0} S_i$, where $S_\infty = \bigcup_{i\geq 0} \bigcap_{j\geq i} S_j$ is the set of persistent clauses (the limit of the derivation), $I_e(S)$ is the set of clauses that can be inferred from S in one step by I_e, and $R(S)$ is the set of clauses that are redundant in S according to R.*

We refer to Bachmair and Ganzinger (1992) for the definition of redundancy criterion. Intuitively, redundant clauses are those that can be deleted by contraction without detriment for the refutation. In this paper we apply Definition 5.1 to the derivations of a strategy $\mathcal{C}$ with expansion rules I_e and redundancy criterion R in the sense that the clauses deleted by the contraction rules of $\mathcal{C}$ are redundant according to R. We shall use two properties of redundancy criteria given in Bachmair and Ganzinger (1992): a redundancy criterion is *monotonic*, that is, if $S \subseteq S'$, then $R(S) \subseteq R(S')$, and redundant clauses are *irrelevant* to establishing the redundancy of other clauses: if $(S' - S) \subseteq R(S')$, then $R(S') \subseteq R(S)$.

For distributed derivations, S_∞ is $\bigcup_{k=0}^{n-1} S_\infty^k$, where S_∞^k is $\bigcup_{i\geq 0} \bigcap_{j\geq i} S_j^k$. Limits for the other components of a distributed derivation may be defined in the same way. Definition 5.1 considers only clauses in S. In a distributed derivation, each process performs expansion inferences from premises in $S \cup V$ and deletes by contraction clauses redundant with respect to $S \cup V$. The following lemma and theorem will bridge this gap. We start by showing that if a clause is redundant with respect to $(S \cup V)_\infty$, then it is redundant with respect to S_∞:

Lemma 5.2. *For all k, $0 \leq k \leq n-1$, $R((S \cup V)_\infty^k) \subseteq R(S_\infty)$.*

Proof. We prove that $R((S \cup V)_\infty) \subseteq R(S_\infty)$: since $(S \cup V)_\infty^k \subseteq (S \cup V)_\infty$, it follows $R((S \cup V)_\infty^k) \subseteq R((S \cup V)_\infty) \subseteq R(S_\infty)$ by monotonicity of R.

If $(S \cup V)_\infty \subseteq S_\infty$ holds, then $R((S \cup V)_\infty) \subseteq R(S_\infty)$ follows by monotonicity of R.

If $(S \cup V)_\infty \subseteq S_\infty$ does not hold, then there exists some clause φ, such that $\varphi \in (S \cup V)_\infty$ but $\varphi \notin S_\infty$. We show that such a clause must be redundant. Since $\varphi \notin S_\infty$, there exist k and n such that $\varphi \in V_n^k$. By Lemma 5.1, $\varphi \in S_i^j$ for some process p_j and stage i. Since $\varphi \notin S_\infty$, φ is deleted by contraction at p_j, that is, φ is redundant: $\varphi \in R((S \cup V)_\infty^j)$.

By monotonicity of R, $(S \cup V)^j_\infty \subseteq (S \cup V)_\infty$ implies that $\varphi \in R((S \cup V)_\infty)$. Thus, we have that every clause that is in $(S \cup V)_\infty$, but not in S_∞, is in $R((S \cup V)_\infty)$. In other words, we have $((S \cup V)_\infty - S_\infty) \subseteq R((S \cup V)_\infty)$. By the second property of redundancy criteria (irrelevance of redundant clauses), it follows that $R((S \cup V)_\infty) \subseteq R(S_\infty)$. □

The theorem shows that if the derivations at the nodes are locally fair on $S \cup V$, and the communication scheme satisfies additional conditions, then the distributed derivation is globally fair:

Theorem 5.1. *If a distributed derivation* $T^k_0 \vdash_{\mathcal{C}} T^k_1 \vdash_{\mathcal{C}} \dots T^k_i \vdash_{\mathcal{C}} \dots$ *is such that*

1 *all raw clauses and all messages are processed:*
$\forall k$, $0 \leq k \leq n-1$, $CP^k_\infty = MI^k_\infty = MO^k_\infty = \emptyset$,

2 *all persistent, non-redundant residents are diffused:*

  (a) $\forall \psi \in (S_\infty - R(S_\infty))$, *there exist process* p_k *and stage* i, $i \geq 0$, *such that* $\psi \in MO^k_i$,

  (b) $\forall \psi \in (S_\infty - R(S_\infty))$, *if* $\psi \in MO^k_i$ *for some* k *and* i, *then for all processes* p_j, $0 \leq j \neq k \leq n-1$, *there exists a stage* l_j, $l_j \geq 0$, *such that* $\varphi \in MI^j_{l_j}$,

3 *all expansion inferences from persistent, non-redundant clauses at any given node* p_k *will be considered either by process* p_k *or by others; in particular, inferences between persistent, non-redundant residents will be considered by* p_k *itself:*
$\forall k$, $0 \leq k \leq n-1$, $I_e((S \cup V)^k_\infty - R((S \cup V)^k_\infty)) \subseteq \bigcup_{i \geq 0} \bigcup_{j=0}^{n-1} CP^j_i$ *and*
$I_e(S^k_\infty - R((S \cup V)^k_\infty)) \subseteq \bigcup_{i \geq 0} CP^k_i$,

then the distributed derivation is uniformly fair*:* $I_e(S_\infty - R(S_\infty)) \subseteq \bigcup_{i \geq 0} \bigcup_{k=0}^{n-1} CP^k_i$.

Proof. Let φ be any clause in $I_e(S_\infty - R(S_\infty))$ with parents $\psi_1, \psi_2 \in S_\infty - R(S_\infty)$. Let p_k and p_h, $0 \leq k, h \leq n-1$, be the processes that own ψ_1 and ψ_2 respectively, that is, $\psi_1 \in S^k_\infty - R(S_\infty)$ and $\psi_2 \in S^h_\infty - R(S_\infty)$.

If $k = h$, then $\varphi \in I_e(S^k_\infty - R(S_\infty))$. By Lemma 5.2, $R((S \cup V)^k_\infty)) \subseteq R(S_\infty)$ and thus $(S^k_\infty - R(S_\infty)) \subseteq (S^k_\infty - R((S \cup V)^k_\infty))$, so that $\varphi \in I_e(S^k_\infty - R((S \cup V)^k_\infty))$. By Condition 3, we have $\varphi \in \bigcup_{i \geq 0} CP^k_i$.

If $k \neq h$, by Condition 2a, we have $\psi_1 \in MO^r_{i_1}$ for some process r and stage i_1 and $\psi_2 \in MO^q_{i_2}$ for some process q and stage i_2. Since $MO_\infty = \emptyset$ by Condition 1, the messages ψ_1 and ψ_2 are broadcast. By Condition 2a, ψ_1 arrives at p_h and ψ_2 arrives at p_k: $\psi_1 \in MI^h_{j_1}$ for some stage j_1 and $\psi_2 \in MI^k_{j_2}$ for some stage j_2. Since $MI_\infty = \emptyset$ by Condition 1, we have that $\psi_1 \in V^h_{l_1}$ for some stage l_1 and $\psi_2 \in V^k_{l_2}$ for some l_2. Since ψ_1 and ψ_2 are persistent, they will not be deleted by backward contraction: $\psi_1 \in V^h_\infty$ and $\psi_2 \in V^k_\infty$. Since they are non-redundant, we have $\psi_1, \psi_2 \in ((S \cup V)^k_\infty - R(S_\infty))$ at node p_k and $\psi_1, \psi_2 \in ((S \cup V)^h_\infty - R(S_\infty))$ at node p_h. By Lemma 5.2 applied as above, we have $\psi_1, \psi_2 \in ((S \cup V)^k_\infty - R((S \cup V)^k_\infty))$ at node p_k and $\psi_1, \psi_2 \in ((S \cup V)^h_\infty - R((S \cup V)^h_\infty))$ at node p_h. By Condition 3, applied to either p_k or p_h, we have $\varphi \in \bigcup_{i \geq 0} \bigcup_{k=0}^{n-1} CP^k_i$. □

Given a specific Clause-Diffusion strategy with a refutationally complete inference system, it suffices to verify the hypotheses of this theorem to establish that the strategy is fair, and thus complete. Condition 1 and 2 express the fairness requirements for the communication scheme[†], while Condition 3 expresses the local fairness of the search plan(s) controlling the inferences at the nodes.

6. Reconstruction of Distributed Proofs

In this section we prove that Modified Clause-Diffusion guarantees proof reconstruction. The first step is to generalize to distributed strategies the notion of *unambiguous* naming scheme:

Definition 6.1. *A distributed theorem-proving strategy $\mathcal{C}$ has an* unambiguous *naming scheme* (A, R) *if, for all derivations,* $T_0^k \vdash_{\mathcal{C}} T_1^k \vdash_{\mathcal{C}} \ldots T_i^k \vdash_{\mathcal{C}} \ldots$, *for all processes* p_k, *for* $0 \leq k \leq n-1$, R *is a bijective function* $R\colon A \rightarrow \bigcup_{i \geq 0} S_i^k \cup V_i^k \cup D_i^k$.

The co-domain of the retrieval function is $S \cup V \cup D$, because these are the components a process will consult when reconstructing the proof, since for a fair strategy $CP_\infty^k = MI_\infty^k = MO_\infty^k = \emptyset$.

The second step is to give requirements for the communication scheme. Condition 2 for fairness says that all persistent non-redundant residents will be diffused. This is not sufficient, however, for proof reconstruction, because the proof may contain non-persistent clauses or persistent but redundant clauses. Thus, we need to require that all *premises* will be broadcast eventually:

Definition 6.2. *A distributed theorem-proving strategy $\mathcal{C}$ has a* comprehensive *communication scheme if, for all derivations,* $T_0^k \vdash_{\mathcal{C}} T_1^k \vdash_{\mathcal{C}} \ldots T_i^k \vdash_{\mathcal{C}} \ldots$, *for all processes* p_k, $0 \leq k \leq n-1$, *if there is a stage* i, $i \geq 0$, *where the search plan* Σ_k *selects* φ *as premise, then there exist a process* p_j, $0 \leq j \leq n-1$ *(possibly, but not necessarily* $j = k$*) and a stage* l, $l \geq 0$, *such that* $\varphi \in MO_l^j$.

One could give a stronger requirement, asking that premises be broadcast before their descendants. However, we shall see that this definition, combined with others, is sufficient. We prefer to give a weaker requirement, so that our treatment is more general. For instance, this definition does not exclude a communication scheme that is comprehensive thanks to a round of post-processing, with ad hoc communication for proof reconstruction. Modified Clause-Diffusion, on the other hand, achieves proof reconstruction by using the communication that is already in place for inferences.

The complementary requirement is that all broadcast clauses will be received by all nodes:

Definition 6.3. *A distributed theorem-proving strategy $\mathcal{C}$ has a* safe *communication scheme if, for all derivations,* $T_0^k \vdash_{\mathcal{C}} T_1^k \vdash_{\mathcal{C}} \ldots T_i^k \vdash_{\mathcal{C}} \ldots$, *for all processes* p_k, $0 \leq k \leq n-1$, *if* $\varphi \in MO_i^k$ *for some stage* i, $i \geq 0$, *then for all processes* p_j, $0 \leq j \neq k \leq n-1$, *there exists a stage* l_j, $l_j \geq 0$, *such that* $\varphi \in MI_{l_j}^j$.

[†] We recall that $CP_\infty^k = \emptyset$ does not mean that CP^k will be empty eventually (which for an infinite derivation may never occur), but that no clause will persist in CP^k, i.e. all clauses added to CP^k will be deleted or moved to other components eventually.

We remark that a communication scheme that allows interleaving of backward contraction and communication may not be safe. Consider, for instance, a communication scheme where broadcasting is implemented by receive-and-forward, and nodes may reduce received messages and forward their reduced forms. Such a scheme may satisfy Condition 2b for fairness, because the latter is only concerned with persistent and non-redundant clauses, which will not be reduced. But it is not safe, because inference messages carrying non-persistent clauses may not be received in the form they were sent. On the other hand, a communication scheme where a message is broadcast in one hop, with no forwarding by intermediate nodes, is safe. Also a receive-and-forward mechanism is safe, if backward contraction is not mingled with receive-and-forward. This is a reasonable constraint, since the end receiver of an inference message will most likely be able to perform the backward contraction steps that the intermediate nodes would perform on the message. Furthermore, interleaving of backward contraction and receive-and-forward means that the broadcast operation is not atomic with respect to the inferences. This makes the design more complicated and less realistic, since in most software systems for programming distributed computations the communication operations, including broadcast, are available to the programmer as primitives.

The following theorem summarizes all the conditions for proof reconstruction:

THEOREM 6.1. *Given a distributed theorem-proving strategy $\mathcal{C}$ such that*

1 *$\mathcal{C}$ has an unambiguous naming scheme,*
2 *$\mathcal{C}$ satisfies the hypotheses of Theorem 5.1 for uniform fairness and*
3 *$\mathcal{C}$ has a comprehensive and safe communication scheme,*

then for all derivations $\mathcal{D}$ in the form $T_0^k \vdash_{\mathcal{C}} T_1^k \vdash_{\mathcal{C}} \ldots T_i^k \vdash_{\mathcal{C}} \ldots$, if process p_i, for some i, $0 \leq i \leq n-1$, generates the empty clause at stage h_i and every process p_k, for all k, $0 \leq k \leq n-1$, terminates at stage h_k, then p_i can reconstruct $at_{\mathcal{D}}(\Box)$ from its final state $(S; V; CP; MI; MO; D)^i_{h_i}$.

PROOF. Since the naming scheme is unambiguous, it is sufficient to show that all clauses in $at_{\mathcal{D}}(\Box)$ are in $S^i_{h_i} \cup V^i_{h_i} \cup D^i_{h_i}$: if they are available, p_i will retrieve them unambiguously. The proof is by induction on the depth m of $at_{\mathcal{D}}(\Box)$.

Base: if $m = 1$, then $at_{\mathcal{D}}(\Box)$ has $\Box$ as root with children the input clauses $\psi_1, \ldots \psi_r$, and p_i generates $\Box$ in one step from $\psi_1, \ldots \psi_r$ at stage h_i. Thus, $\psi_1, \ldots \psi_r$ are in $S^i_{h_i} \cup V^i_{h_i}$.

Induction hypothesis: all clauses in $at_{\mathcal{D}}(\Box)$ up to depth $m = q$ are in $S^i_{h_i} \cup V^i_{h_i} \cup D^i_{h_i}$.

Induction step: let φ be a clause at depth q in $at_{\mathcal{D}}(\Box)$ and let $\psi_1, \ldots \psi_r$ be its parents at depth $q+1$ (This proof applies regardless of whether the step generating φ from $\psi_1, \ldots \psi_r$ is an expansion or a contraction step.). We need to consider the following cases:

1 The step generating φ from $\psi_1, \ldots \psi_r$ was executed at p_i at some stage l_i, $0 \leq l_i < h_i$. This means that $\psi_1, \ldots \psi_r \in (S \cup V)^i_{l_i}$.

   (a) If $\psi_1, \ldots \psi_r$ are all persistent, then $\psi_1, \ldots \psi_r \in (S \cup V)^i_{h_i}$. (This subcase applies only if the step generating φ from $\psi_1, \ldots \psi_r$ is an expansion step.)

   (b) If $\psi_1, \ldots \psi_r$ are not all persistent, then there is some ψ_j, $1 \leq j \leq r$, which was deleted by p_i. Since ψ_j was in $S \cup V$, it must have been deleted by backward contraction. Since the strategy saves in D the clauses deleted by backward contraction, $\psi_j \in D^i_{h_i}$ and $\psi_1, \ldots \psi_r \in S^i_{h_i} \cup V^i_{h_i} \cup D^i_{h_i}$.

2 The step generating φ from $\psi_1, \ldots \psi_r$ was executed at some p_k, $k \neq i$. Since the strategy has a comprehensive communication scheme and $\psi_1, \ldots \psi_r$ were used as premises, $\psi_1, \ldots \psi_r$ were broadcast. Since the communication scheme is also safe, they were received by all processes. In particular, they were received by p_i: for all ψ_j, $1 \leq j \leq r$, there is a stage l_j, $0 \leq l_j < h_i$, such that $\psi_j \in MI^i_{l_j}$. By hypothesis 1 of Theorem 5.1, $MI^i_\infty = MI^i_{h_i} = \emptyset$. Thus, $\psi_1, \ldots \psi_r$ are moved from the MI component to $S \cup V$. For all j, $1 \leq j \leq r$, if ψ_j is persistent, then $\psi_j \in (S \cup V)^i_{h_i}$. If ψ_j is not persistent, then, since it is in $S \cup V$, it must have been deleted by backward contraction, and we have $\psi_j \in D^i_{h_i}$.

□

Modified Clause-Diffusion has an unambiguous naming scheme (Section 5.1) and a comprehensive communication scheme, because it takes the eager approach of broadcasting clauses right after forward contraction (Section 5.2). If, in addition, the communication scheme is safe, and the specific strategy is fair, then proof reconstruction is guaranteed.

7. Discussion

We have studied the problem of proof reconstruction in the context of distributed theorem proving by concurrent, deductive, peer processes, with asynchronous communication and distributed memory. The proof reconstruction problem consists in guaranteeing that the successful process is able to reconstruct the distributed proof based solely on the final state of its database. We showed that this property is not trivial, as the successful process may fail to find locally all the clauses that are necessary to reconstruct the proof, even if the distributed strategy is fair and complete.

As a starting point, we assumed the methodology for distributed deduction by Clause-Diffusion that we developed in previous work. By analyzing the possible failures of proof reconstruction, we focused on the components of a strategy that are relevant to the reconstruction of proofs: the communication scheme, the naming scheme and the treatment of the raw clauses generated by backward contraction. Based on this analysis, we proposed a Modified Clause-Diffusion method, we proved that it is fair, thus complete, and guarantees proof reconstruction. This result showed that proof reconstruction can be achieved in distributed theorem proving with distributed memory, peer processes and asynchronous communication, without adding centralized control or ad hoc post-processing, and using solely the communication already prescribed by the method for the distribution of inferences. In addition to being a desirable property, we feel that proof reconstruction led us to polish and streamline Clause-Diffusion significantly.

We have implemented Modified Clause-Diffusion in the prototype *Peers-mcd*, a new version of *Peers* (Bonacina and McCune 1994). Like its predecessor, Peers-mcd features contraction-based strategies for equational problems, possibly with AC operators. Peers-mcd succeeded in reconstructing the proof in all experiments, according to the theoretical results. Table 1 reports some experiments with Peers-mcd on a local area network of HP workstations: n-Peers is Peers with n nodes, where the first and second nodes are HP 715/80, the third, fourth and fifth are HP 715/75, the sixth is an HP 715/50 and the seventh is an HP 715/33. All nodes have 64M of memory, except the seventh which has 32M of memory. The run-time of n-Peers is the CPU time (in sec.) of the first process

Table 1. Experiments with Peers-mcd.

Problem	*1-Peers*	*2-Peers*	*4-Peers*	*6-Peers*	*7-Peers*
kbcomm	7.38	1.55	1.00	0.78	0.44
x3	92.80	20.26	24.58	12.97	15.95
r2	14.16	20.74	7.74	9.68	6.68
r14	154.03	36.08	96.63	16.33	61.41
s12	54.51	15.59	11.51	25.33	24.03
s32	7.18	2.39	3.66	4.40	2.95

to succeed. The other processes run till either they receive a halting message or also find a proof, whichever happens first.

Problem *kbcomm* is the commutator problem in group theory, $x3$ is the problem of proving that $x^3 = x$ implies commutativity in ring theory, $r2$ is the problem in Robbins algebra called *Robbins* in Lusk and McCune (1992), $r14$ is a related problem (courtesy of Bill McCune), $s12$ and $s32$ are problems in algebraic logic (courtesy of Anita Wasilewska and Jieh Hsiang).

References

Avenhaus, J., Denzinger, J. (1993). Distributing equational theorem proving. In Kirchner, C., editor, *Fifth Conf. on Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 62–76, Montréal, Canada, June. Springer Verlag.

Bachmair, L., Dershowitz, N., Plaisted, D. A. (1989). Completion without failure. In Ait-Kaci, H., Nivat, M., editors, *Resolution of Equations in Algebraic Structures - Rewriting Techniques*, volume 2, pages 1–30, New York. Academic Press.

Bachmair, L., Ganzinger, H. (1992). Non-clausal resolution and superposition with selection and redundancy criteria. In Voronkov, A., editor, *Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 273–284. Springer Verlag.

Bonacina, M. P. (1994). On the reconstruction of proofs in distributed theorem proving with contraction: a modified Clause-Diffusion method. In Hong, H., editor, *First Int. Symp. on Parallel Symbolic Computation*, volume 5 of *Lecture Notes Series in Computing*, pages 22–33, Linz, Austria, September. World Scientific.

Bonacina, M. P., Hsiang, J. (1994). Parallelization of deduction strategies: an analytical study. *J. Automated Reasoning*, 13:1–33.

Bonacina, M. P., Hsiang, J. (1995a). The Clause-Diffusion methodology for distributed deduction. *Fundamenta Informaticae*, 24:177–207.

Bonacina, M. P., Hsiang, J. (1995b). Distributed deduction by Clause-Diffusion: the Aquarius prover. *J. Symbolic Computation*, 19:245–267.

Bonacina, M. P., McCune, W. W. (1994). Distributed theorem proving by Peers. In Bundy, A., editor, *Twelfth Conf. on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 841–845, Nancy, France, June. Springer Verlag.

Denzinger, J., Schulz, S. (1994). Recording, analyzing and presenting distributed deduction processes. In Hong, H., editor, *First Int. Symp. on Parallel Symbolic Computation*, volume 5 of *Lecture Notes Series in Computing*, pages 114–123, Linz, Austria, September. World Scientific.

Hsiang, J., Rusinowitch, M. (1987). On word problems in equational theories. In Ottman, Th., editor, *Fourteenth Int. Conf. on Automata, Languages and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 54–71, Karlsruhe, Germany, July. Springer Verlag.

Lusk, E. L., McCune, W. W. (1992). Experiments with ROO: a parallel automated deduction system. In Fronhöfer, B., Wrightson, G., editors, *Parallelization in Inference Systems*, volume 590 of *Lecture Notes in Artificial Intelligence*, pages 139–162. Springer Verlag.

McCune, W. W. (1994). Otter 3.0 reference manual and guide. Technical Report 6, Mathematics and Computer Science Division, Argonne National Laboratory.

Suttner, C. B., Schumann, J. (1994). Parallel automated theorem proving. In Kanal, L., Kumar, V., Kitano, H., Suttner, C. B., editors, *Parallel Processing for Artificial Intelligence*. Elsevier, Amsterdam.