On the reconstruction of proofs in distributed theorem proving with contraction: a modified Clause-Diffusion method

Maria Paola Bonacina *
Department of Computer Science
University of Iowa
Iowa City, IA 52242-1419, USA
bonacina@cs.uiowa.edu

Abstract: Proof reconstruction is the operation of extracting the computed proof from the trace of a theorem proving run. In experiments with distributed theorem proving by Clause-Diffusion, we observed that proof reconstruction is far from being a trivial task in distributed theorem proving with contraction: because of the distributed nature of the derivation and especially because of backward 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 present a modified Clause-Diffusion method 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 and without resorting to a round of post-processing with additional, ad hoc communication. We define a set of sufficient conditions and we prove that if a distributed strategy satisfies these requirements, then proof reconstruction is guaranteed. We show how the modified Clause-Diffusion method satisfies these requirements and we compare the modified Clause-Diffusion method with previous versions of Clause-Diffusion and related approaches.

Keywords: Automated theorem proving, distributed deduction, contraction.

1 Introduction

Proof reconstruction is an important feature of theorem provers that implement resolution-based or completion-based strategies for clausal and equational refutational theorem proving. These strategies work primarily by forward reasoning, that is, by deriving consequences from the axioms and the negation of the input theorem, until a contradiction, the empty clause, is generated. While searching for the empty clause, these procedures typically generate 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 are not regarded as exceptional. Since an output of this size is unpractical for most purposes, theorem provers incorporate an algorithm to reconstruct the computed proof, that is, to extract from the record of all the generated clauses those that are related by inference steps to the empty clause. The task of extracting a computed proof from a much larger set of inference steps does not pertain solely to forward reasoning procedures. The need of shrinking a huge output to a smaller, more manageable, more meaningful output is typical for all problems involving *search*, because solving a problem by searching consists in generating many possibilities until a successful one is found.

In this paper, we focuse on proof reconstruction in *distributed theorem proving* by forward

*Supported in part by the GE Foundation Faculty Fellowship to the University of Iowa and by the National Science Foundation with grant CCR-94-08667.

reasoning procedures. 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 data base 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 [6] and [11] for surveys on parallel and distributed deduction for different strategies and architectures and to [5] for our 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 data base. 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 data base of p_j. We observed this and other difficulties during experiments with our Clause-Diffusion prototypes Aquarius [4] and Peers [7].

In this work we give a systematic treatment of the problem of proof reconstruction in distributed theorem proving and we propose a solution. First, we define formally proof reconstruction. Second, we overview briefly the Clause-Diffusion approach. Then we classify the failures in distributed proof reconstruction reported in our experiments. These observations guide the design of the modified Clause-Diffusion method, that we describe along with the proof reconstruction failures, showing informally how modified Clause-Diffusion prevent them. In the formal sections of the paper, we prove that the modified Clause-Diffusion method is fair, and therefore complete, if the underlying inference system is complete, and that it guarantees proof reconstruction. We demonstrate this property by formulating sufficient conditions for distributed proof reconstruction: we show that our conditions are sufficient and that the modified Clause-Diffusion method fulfills them. Modified Clause-Diffusion guarantees proof reconstruction by using the asynchronous communication that is already in place in the Clause-Diffusion methodology for the distribution of inferences. 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 simpler and more uniform schemes for communication, identification and allocation of clauses, that are executed in a purely distributed, asynchronous fashion by the processes.

2 Proof reconstruction

This section contains preliminary material: a definition of the computed proof and the conditions for proof reconstruction in sequential theorem proving. We assume to have a strategy $\mathcal{C} =< I; \Sigma >$ for first-order clausal theorem proving, where I is the set of inference rules and Σ is the search plan controlling the application of the inference rules. Given a theorem proving problem in refutational clausal form $S_0 = S \cup \{\neg\varphi\}$, the strategy $\mathcal{C}$ will generate a derivation

$S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \ldots S_i \vdash_{\mathcal{C}} S_{i+1} \vdash_{\mathcal{C}} \ldots$,

where at each step an inference rule is applied to selected premises according to the search plan. The proof computed by a derivation can be defined and represented graphically by adopting the classical convention of representing proofs as trees. An expansion inference step, such as a resolution or paramodulation step, deriving clause φ_n from premises $\varphi_1, \ldots \varphi_{n-1}$, is represented as a subtree with root φ_n and children $\varphi_1, \ldots \varphi_{n-1}$. We denote it by the pair $(\varphi_n, f(\varphi_1, \ldots, \varphi_{n-1}))$, where f is the symbol of the applied inference

rule. This representation can be applied also to contraction inferences, such as simplification or normalization, where a clause φ_{n-1} is reduced to φ_n by simplifiers $\varphi_1, \dots \varphi_{n-2}$: φ_n is the root and $\varphi_1, \dots \varphi_{n-2}, \varphi_{n-1}$ are the children. In turn, $\varphi_1, \dots \varphi_{n-1}$ have children, representing their premises. Thus a tree represents a proof of the clause associated to its root:

Definition 2.1 *Given a derivation*
$S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \dots \vdash_{\mathcal{C}} S_i \ \dots$,
for all clauses $\varphi \in \bigcup_{i \geq 0} S_i$, *the* ancestors-tree *of* φ, *denoted by* $at(\varphi)$, *is defined inductively as follows:*

- *if* $\varphi \in S_0$, *i.e.* φ *is an input clause, then* $at(\varphi) = \varphi$,
- *if* φ *is derived at stage* $i > 0$ *by inference rule* f, *with premises* $\varphi_1, \dots \varphi_{n-1}$, *then*

$$at(\varphi) = (\varphi, f(at(\varphi_1), \dots at(\varphi_{n-1}))).$$

The ancestors-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. Since each clause has its own set of variables, the same clause is never derived twice. Rather, multiple *variants* of a clause, i.e. clauses that differ only by a permutation of variables, are typically generated during a derivation. If a clause φ is derived at stage i and a variant φ' of φ is derived at stage j, their ancestors-trees are two distinct objects, even if they may represent the same inferences logically. We recall that treating variants as distinct clauses is necessary to represent proofs as trees: if variants are not differentiated, the graph of the proof may contain cycles. Also, a proof tree may have more than one node labelled by the same clause, since a clause may be a premise of more than one step in a proof.

Definition 2.2 *Given a successful derivation*
$S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \dots S_i \vdash_{\mathcal{C}} S_{i+1} \ \dots \vdash_{\mathcal{C}} S_h$,
the computed proof *is* $at(\square)$, *the ancestors-tree of the empty clause.*

We call *naming scheme* the mechanism that a theorem prover uses to associate *identifiers* to clauses. Identifiers are chosen from a domain A, which is ordered and has the cardinality of $\mathbb{N}$, often $\mathbb{N}$ itself. A naming scheme induces a *retrieval relation* $R \subseteq A \times \mathcal{L}$, where $\mathcal{L}$ is the language of clauses on the given signature, and $(x, \varphi) \in R$ means that x is the identifier of clause φ. We denote by (A, R) the naming scheme that uses domain A and induces retrieval relation R. The relevant property for proof reconstruction is the following:

Definition 2.3 *A theorem proving strategy* $\mathcal{C}$ *has an* unambiguous *naming scheme* (A, R) *if, for all derivations by* $\mathcal{C}$, $S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \dots S_i \vdash_{\mathcal{C}} S_{i+1} \ \dots$, $R\colon A \to \bigcup_{i \geq 0} S_i$ *is a bijective function.*

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

Existing sequential theorem provers implement proof reconstruction by using unambiguous naming schemes. For instance, the theorem prover Otter [10] 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 (expansion or contraction) that derived it. When an empty clause $\square$ is generated, the prover reconstructs $at(\square)$ by retrieving the parents of $\square$, then the parents of the parents and so on, until the reconstruction process retrieves clauses that were part of the input set. This algorithm assumes that the chains of references among clauses contain no cycles, (i.e. the prover builds proof trees correctly), and information about the used inference rules is stored with the clauses, so that proof reconstruction reduces to ancestors retrieval. We shall also make these assumptions in this paper.

Additional care is needed if the strategy features contraction inference rules, such as simplification, because clauses deleted by simplification may appear in $at(\square)$ and thus should be saved for the purpose of proof reconstruction even if they are no longer used for inferences. Saving all clauses deleted by simplification is not appealing because of memory consumption. Fortunately, it is possible to establish a priori that a large set of clauses deleted by contraction will not be needed for proof reconstruction. The key observation is the distinction between *forward contraction*, the contraction of newly generated clauses right after generation, and *backward contraction*, the contraction of all other clauses [6]. Clauses deleted by forward contraction are not used as premises of other steps before deletion and therefore will not be needed for proof reconstruction. Clauses deleted by backward contraction may be needed for proof reconstruction, because they may have been used as premises of other steps before being deleted. Therefore, it is sufficient to save the

clauses deleted by backward contraction in a separate component D of the data base, which will be consulted only by the proof reconstruction algorithm:

$(S_0; D_0) \vdash_{\mathcal{C}} (S_1; D_1) \vdash_{\mathcal{C}} \ldots (S_i; D_i) \vdash_{\mathcal{C}} \ldots$,

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

3 Modified Clause-Diffusion

In this section, first we give a brief overview of the general elements of the Clause-Diffusion approach [5] and then we concentrate on proof reconstruction and modified Clause-Diffusion. We classify the types of failure in proof reconstruction that we observed experimentally and we show informally how modified Clause-Diffusion prevents such failures.

Clause-Diffusion seeks to realize a form of coarse-grain parallelism for theorem proving called *parallelism at the search level*. Analysis and motivation for this choice were given in [6]. The idea is to have concurrent deductive processes $p_0, \ldots p_{n-1}$ 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

$p_0, \ldots p_{n-1}$

runs on a node of the system, also denoted by $p_0, \ldots p_{n-1}$. Given a theorem proving strategy $\mathcal{C}$ and an input problem S_0, each process p_k executes $\mathcal{C}$, generating its derivation

$S_0^k \vdash_{\mathcal{C}} S_1^k \vdash_{\mathcal{C}} \ldots S_i^k \ldots$.

The distributed derivation is formed by the collection of these derivations and it succeeds as soon as one of them does. The set S^k represents the local data base at p_k, whereas $\bigcup_{k=0}^{n-1} S^k$ represents the global data base. The processes may execute different versions of $\mathcal{C}$, e.g. differing in the search plan.

Intuitively, this approach is most fruitful, if the processes search different portions of the search space. Clause-Diffusion tries to realize this effect by assigning each clause to a process and establishing that each process will perform only those inferences that involve its own clauses, called its *residents*. Consider for instance the paramodulation inference rule. According to Clause-Diffusion, process p_k will perform only those paramodulation steps that paramodulate into a clause belonging to 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 other expansion inference rules, such as resolution.

In order to make inferences between clauses belonging to different processes possible, a Clause-Diffusion method features a *communication scheme*, according to which the processes broadcast, or *diffuse* (hence the name of the methodology) their clauses. The messages carrying clauses are called *inference messages*. It follows that the data base of each process contains both residents and non-resident clauses that were received as inference messages. By collecting clauses received as messages, each process forms an approximated version, termed *localized image set*, of the current state of the global data base. This set is used as a set of simplifiers, so that a process can contract a clause with respect to the global data base, not just its own local data base. This feature is called *distributed global contraction*.

The issue of subdividing a theorem proving problem is a difficult one: since a clause can be generated in many ways, subdividing the clauses among the processes does not prevent the processes from generating common clauses. The data-driven partition of inferences in Clause-Diffusion provides an approximation to the ideal of a true partition of the search space. Specific Clause-Diffusion strategies may be defined by giving concrete search plans, algorithms for allocation of residents, schemes for communication and distributed global contraction et cetera. We refer to [5, 3, 4, 7] for a detailed presentation of the Clause-Diffusion methodology and specific strategies.

3.1 The communication scheme

The scheduling of communication affects proof reconstruction, because proof reconstruction may fail if a clause is broadcast, and received, earlier than one of its ancestors. In turn, this may happen only if clauses are used as premises, and thus have descendants, before being broadcast. Thus, we call this type of failure *failure by delayed diffusion*, because broadcasting is scheduled too late with respect to inferences:

- *Failure by delayed diffusion*: consider clause φ belonging to process p_k. Assume that p_k use φ as premise before broadcasting it. For instance, φ is applied as a simplifier to reduce ψ to ψ'. Thus φ is a parent of ψ'. Depending on how clauses are selected for broadcasting, it may happen that p_k broadcasts ψ' *before* it broadcasts φ. 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 re-

construct the proof, because the reference to φ's identifier in ψ' cannot be solved. It may also happen that p_k applies φ to generate ψ' and then deletes φ by backward contraction, so that p_k does not broadcast φ at all. The effect on proof reconstruction is similarly negative.

The original Clause-Diffusion made each process responsible for broadcasting its residents and specified not to delay broadcasting after expansion. In the provers of [4, 7], each process broadcasts its residents when they are selected for the first time as premises of expansion inferences. Therefore, in our experiments failures by delayed diffusion were due mostly to missing contraction-parents. If clauses are broadcast after expansion, failures by delayed diffusion for expansion-parents may also occur. Modified Clause-Diffusion prevents these failures by adopting an *eager* communication scheme:

- *Broadcast inference message*: each process is responsible for broadcasting as inference messages the clauses it generates. Whenever process p_k generates a new, or *raw*, clause φ, p_k reduces φ to its normal form φ' (forward contraction) and if φ' is not deleted, p_k executes the allocation algorithm (of the specific Clause-Diffusion strategy), to decide which process φ' belongs to. Regardless of whether φ is assigned to p_k, p_k keeps φ and also broadcasts it as an inference message.

Thus a clause will be broadcast by the process that generates it, not by the process that owns it. 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 for the benefit of the other processes. If φ is assigned to another node p_j, then p_k keeps and broadcasts φ, realizing in one operation both the goal of sending φ to its owner p_j and the goal of broadcasting φ to all the processes. Failures by delayed diffusion cannot occur, because clauses are broadcast before being used as premises.

3.2 The naming scheme

A naming scheme for a distributed derivation needs to identify each clause uniquely in the global data base. The naming scheme is related to the communication scheme because a clause needs to receive its global identifier *before* being broadcast. Therefore, the process that broadcasts a clause should also name it. In the original Clause-Diffusion, both tasks are performed by the owner of the clause. In modified Clause-Diffusion, a clause is given its global identifier by the process that generates it:

- the global identifier of a clause φ generated by p_k and allocated to p_j (by the allocation algorithm run at p_k) is $< j, k, l >$, 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 $< k, k, l >$, with the same meaning for l. This naming scheme has *no repetitions*, meaning that no process generates twice the same identifier. Its implementation requires that each process keep a count of how many clauses it has already allocated to any process, including itself.

3.3 The treatment of clauses generated by backward contraction

In a strategy with backward contraction, raw clauses are generated not only by expansion but also by backward contraction. The treatment of raw clauses generated by backward contraction is a critical issue for distributed proof reconstruction and for distributed deduction in general. The Clause-Diffusion methodology recommends that a process broadcast a clause only if it is irreducible with respect to all the simplifiers available to the process, in order to prevent broadcasting a clause which is known to be redundant. Modified Clause-Diffusion respects this indication by broadcasting clauses right after normalization by forward contraction. Of course, a clause that was irreducible at the time of broadcasting may become reducible at a later stage, as other simplifiers are generated. Consider a clause φ that was broadcast at some stage of a distributed derivation, is stored at all the nodes and is reducible to φ'.

If all the processes are allowed to reduce φ to φ' and to treat the generated clause like a raw clause generated by expansion, up to n copies of φ' may be generated, normalized, given a (different) global identifier and broadcast. (We recall that the processes are asynchronous, and therefore unlikely to perform these steps at the same time.) This will induce in turn more backward contraction steps, e.g. subsumption steps, to get rid of all the duplicates. In order to avoid the redundancy of this scenario, one may restrict backward contraction based on ownership of clauses, by establishing that a process may simplify only its own clauses. Accordingly, only the owner of φ reduces it to φ' and broadcasts φ' as a new

clause (we assume here for simplicity that φ' is irreducible). Upon receiving the inference message φ', the other processes will use it to replace φ in their data bases. For this purpose, and for proof reconstruction, φ' would carry in its history the information that it was generated by backward contraction of φ. The disadvantage of this scheme is that the processes are not allowed to delete by simplification those redundant clauses that are not their residents. The effect of backward normalization is delayed until reduced forms are received as inference messages. Since most of the data base of a process may be made eventually of non-resident clauses, the limitation of the contraction power of the processes is significant.

The original Clause-Diffusion method adopted a compromise between unrestricted backward contraction and subdivision of backward contraction by ownership. All processes reduce φ to φ', but only the owner, say p_i, of φ is allowed to generate a new identifier for φ' and broadcast φ' with its new name. The other processes generate φ' and store it temporarily with the identifier of φ. When they receive the inference message φ' broadcast by p_i, they update their data bases with the correct identifier of φ'. A problem with this approach is that it makes the naming scheme ambiguous, because for some time φ and φ' have the same identifier at all nodes different from p_i. Indeed, proof reconstruction may fail in two ways:

1. *Failure by name clash at a receiver*:
   if a process p_k, $k \neq i$, reduces φ to φ', stores φ' with the identifier x which was originally φ's, and later finds a proof including a reference to φ through x, p_k may not be able to reconstruct it correctly, because the identifier x may retrieve φ' instead.

2. *Failure by name clash at the sender*:
   processes other than p_i may use φ' as premise before its identifier is updated. Therefore, the histories of other clauses may contain occurrences of identifier x referring to φ'. If p_i finds a proof containing one such reference, it will not be able to reconstruct it, because the identifier x will retrieve φ at p_i.

This scheme cannot be fixed by allowing the processes other than p_i to generate new names for their copies of φ': the naming scheme would still be ambiguous, since φ' would be stored under different indentifiers and name clashes on occurrences of φ' would occur.

Modified Clause-Diffusion proposes a different compromise, that preserves proof reconstruction. 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 processes the contraction step initiated by deleting φ will be completed with no additional work, when the inference message φ' is received and stored. 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 redundant communication and duplication of unrestricted backward contraction, without strongly reducing the contraction power of the processes, since they can still delete redundant clauses regardless of ownership. Second, generation of new clauses by backward contraction and generation of raw clauses by expansion are treated uniformly: both are restricted based on ownership of clauses; all raw clauses and all inference messages are handled in the same way, regardless of how they were generated. All received inference messages can simply be added to the data base, because it is not necessary to use them to update incorrect identifiers or to replace copies of their ancestors. Finally, this scheme implies that all clauses generated by backward contraction get new identifiers. Together with the assumption that the naming scheme has no repetitions, this means that the naming scheme is unambiguous and there are no name clashes.

3.4 Distributed derivations

We conclude this section on modified Clause-Diffusion by summarizing its operations in a formal description of its derivations. A distributed derivation is made of a collection of n derivations

$T_0^k \vdash_{\mathcal{C}} T_1^k \vdash_{\mathcal{C}} \ldots T_i^k \vdash_{\mathcal{C}} \ldots,$

for $0 \leq k \leq n-1$, by the processes $p_0, p_1, \ldots, p_{n-1}$. In a derivation the superscript indicates the process and the subscript indicates the stage. Here and in the rest of the paper T is the tuple

$(S; V; CP; MI; MO; D)$

where S is the set of residents, V is the set of non-

resident clauses currently held at a node, CP is the set of raw clauses, MI is the set of inference messages being received (input), MO is the set of inference messages to be broadcast (output) and D is the set of clauses deleted by backward contraction. A distributed derivation succeeds as soon as one of its component derivations does. The different types of operations work as follows:

- Expansion takes premises in $S \cup V$ and puts the generated raw clauses in CP. For example, if ψ_1 belongs to p_k and ψ_2 belongs to p_h, we have: $\psi_1 \in S^k$, $\psi_2 \in S^h$, $\psi_1 \in V^h$ and $\psi_2 \in V^k$; p_k paramodulates ψ_2 into ψ_1 and p_h paramodulates ψ_1 into ψ_2.

- Forward contraction applies the simplifiers in $S \cup V$ to normalize the raw clauses in CP: deleted clauses disappear, whereas a non-trivial normal form goes in MO and in S, if it belongs to the process, in V otherwise.

- Backward contraction applies the simplifiers in $S \cup V$ to the clauses in $S \cup V$: deleted clauses go in D, whereas a non-trivial normal form goes in MO and in S, if it belongs to the process, in V otherwise.

- The act of receiving an inference message is represented by the appearance of the message in MI. The receiver processes the received inference message by moving it from MI to S if the clause belongs to the process, to V otherwise.

- The act of broadcasting an inference message is initiated by putting the clause in MO; the effect of broadcasting is represented in the derivation by the clause appearing in the MI components of all the other processes at later stages.

Since each clause added to V is broadcast and added to S by one of the processes, it follows that all clauses in V are copies, or "images", of clauses in S: if $\varphi \in V_i^k$, then $\varphi \in S_j^h$, for some h and j. The union $S^k \cup V^k$ forms the localized image set of process p_k, that is, the "image" of the global data base known to p_k.

3.5 Uniform fairness

Fairness of the original Clause-Diffusion method was proved in [3]. 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 3.1 (Bachmair, Ganzinger 1992) [2] *A derivation*

$S_0 \vdash_{\mathcal{C}} S_1 \vdash_{\mathcal{C}} \ldots S_i \vdash_{\mathcal{C}} S_{i+1} \ldots$

is uniformly fair *if* $I_e(S_\infty - R(S_\infty)) \subseteq \bigcup_{i \geq 0} S_i$, *where* $S_\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} S_j$ *(the limit of the derivation) is the set of persistent clauses,* $I_e(S)$ *is the set of clauses that can be inferred from premises in* S *in one expansion step of the given strategy, and* $R(S)$ *is the set of clauses that are redundant in* S *based on the redundancy criterion* R *of the given strategy.*

We refer to [2] for the definition of redundancy criterion. Intuitively, redundant clauses are those that can be deleted by contraction without detriment for the refutation. For the purpose of this paper we simply need to recall that R is the redundancy criterion of strategy $\mathcal{C}$ in the sense that the clauses deleted by the contraction rules of $\mathcal{C}$ are redundant according to R. Also, we shall use two properties of redundancy criteria given in [2]: a redundancy criterion is *monotonic*, that is, if $S \subseteq S'$, then $R(S) \subseteq R(S')$, and redundant clauses are *irrelevant* in establishing the redundancy of other clauses: if $(S' - S) \subseteq R(S')$, then $R(S') \subseteq R(S)$.

For distributed derivations, we have local and global limits, e.g. $S_\infty^k = \bigcup_{i \geq 0} \bigcap_{j \geq i} S_j^k$ and $S_\infty = \bigcup_{k=0}^{n-1} S_\infty^k$. Limits for the other components of a distributed derivation are defined in the same way. The definition of uniform fairness considers clauses in S only. However, each process in a distributed derivation 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. The lemma will show that if a clause is redundant with respect to $(S \cup V)_\infty$, then it is redundant with respect to S_∞. The theorem will show that if the local derivations are uniformly fair with respect to expansion inferences on $S \cup V$ and the communication scheme satisfies additional conditions, then the distributed derivation is uniformly fair.

Lemma 3.1 *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 $R((S \cup$

$V)^k_\infty) \subseteq R((S \cup V)_\infty)) \subseteq R(S_\infty)$ 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, which is persistent in the V component of the derivation, but not in the S component: $\varphi \in V_\infty$ but $\varphi \notin S_\infty$. We show that such a clause must be redundant. Since each clause added to the V component is broadcast and added to the S component by one of the processes, $\varphi \in S^j_i$ 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: $\varphi \in R((S \cup V)^j_\infty)$. Since $(S \cup V)^j_\infty \subseteq (S \cup V)_\infty$, it follows by monotonicity of R 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 $R((S \cup V)_\infty) \subseteq R(S_\infty)$. □

Theorem 3.1 *If a distributed derivation*
$T^k_0 \vdash_{\mathcal{C}} T^k_1 \vdash_{\mathcal{C}} \ldots T^k_i \vdash_{\mathcal{C}} \ldots$,
with $T = (S; V; CP; MI; MO; D)$, *is such that*

1. *all raw clauses, all incoming messages and all outgoing 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 broadcast:*
   $\forall \psi \in (S_\infty - R(S_\infty))$, *there exist a process* p_k *and a stage* i, $i \geq 0$, *such that* $\psi \in MO^k_i$,
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:*
   $\forall k,\ 0 \leq k \leq n-1,$
   $I_e((S \cup V)^k_\infty - R((S \cup V)^k_\infty)) \subseteq \bigcup_{i \geq 0} CP_i$
   where $CP_i = \bigcup_{j=0}^{n-1} CP^j_i$
   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*:* $I_e(S_\infty - R(S_\infty)) \subseteq \bigcup_{k=0}^{n-1} \bigcup_{i \geq 0} CP^k_i$.

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 3.1, $R((S \cup V)^k_\infty)) \subseteq R(S_\infty)$ 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 2, 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 diffused. Since ψ_1 and ψ_2 are persistent and non-redundant, they cannot be contracted on the way and therefore 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 3.1 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 $\varphi \in \bigcup_{k=0}^{n-1} \bigcup_{i \geq 0} CP^k_i$. □

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. For condition 1, we emphasize that based on the definition of limit, $CP^k_\infty = \emptyset$ does not mean that CP^k will be eventually empty, 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 removed eventually.

4 Reconstruction of distributed proofs

In this section we develop formally the discussion of distributed proof reconstruction of Sections 3.1 and 3.3. The first step is to generalize to distributed strategies the notion of *unambiguous* naming scheme:

Definition 4.1 *A distributed theorem proving strategy* $\mathcal{C}$ *has a* globally unambiguous *naming scheme* (A, R) *if, for all derivations,*
$T^k_0 \vdash_{\mathcal{C}} T^k_1 \vdash_{\mathcal{C}} \ldots T^k_i \vdash_{\mathcal{C}} \ldots$,
with $T = (S; V; CP; MI; MO; D)$, *for all processes* p_k, *for* $0 \leq k \leq n-1$, R *is a bijective function* $R\colon A \to \bigcup_{i \geq 0} S^k_i \cup V^k_i \cup D^k_i$.

The codomain of the retrieval function is given by $S \cup V \cup D$, because these are the components a process will consult when reconstructing the proof.

Condition 2 for fairness says that all persistent non-redundant residents will be broadcast. 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 the communication scheme of the distributed strategy satisfies an additional property:

Definition 4.2 *A distributed theorem proving strategy $\mathcal{C}$ has a* comprehensive *communication scheme if, for all derivations,*

$T_0^k \vdash_{\mathcal{C}} T_1^k \vdash_{\mathcal{C}} \ldots T_i^k \vdash_{\mathcal{C}} \ldots$,

with $T = (S; V; CP; MI; MO; D)$, 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 $\varphi \in MO_l^j$.

In other words, all premises will be broadcast eventually. This definition is the weakest requirement that is sufficient to guarantee proof reconstruction. We chose to give the weakest requirement, so that our treatment is as general as possible. Accordingly, 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. For Clause-Diffusion strategies, however, the primary goal of the communication scheme is not proof reconstruction but the distribution of inferences. Therefore, it is much more interesting, and more efficient, to achieve proof reconstruction by using the communication that is already in place for inferences, as we do in modified Clause-Diffusion.

Definitions are given for infinite derivations for the sake of generality, but for the purpose of proof reconstruction, we are interested in successful, and therefore finite, derivations. For finite derivations, Definition 4.2 says that all premises will be broadcast before termination. Termination may be implemented by having the successful process broadcasting a halting message, that reaches the other processes like an asynchronous interrupt and forces them to halt. In such a context, the earlier the processes broadcast their clauses, the better is the communication scheme for the purpose of implementing Definition 4.2. A communication scheme that delays broadcasting till after the clauses have been used as premises may fail to be comprehensive and cause failures by delayed diffusion (see Section 3.1), because a process may halt upon receiving a halting message before having broadcast all the clauses it used as premises. As we discussed in Section 3.1, modified Clause-Diffusion takes the eager (and safest) approach of broadcasting new clauses as soon as possible, right after forward contraction.

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

Definition 4.3 *A distributed theorem proving strategy $\mathcal{C}$ has a* safe *communication scheme if, for all derivations,*

$T_0^k \vdash_{\mathcal{C}} T_1^k \vdash_{\mathcal{C}} \ldots T_i^k \vdash_{\mathcal{C}} \ldots$,

with $T = (S; V; CP; MI; MO; D)$, 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 remark that a communication scheme which allows unrestricted interleaving of backward contraction and communication may not be safe. Consider, for instance, a communication scheme where broadcasting is implemented by a receive-and-forward mechanism along routes of more than one hops. Moreover, assume that the intermediate nodes are allowed to apply backward contraction to received messages before forwarding them and to forward their reduced forms. Such a scheme can be fair, because fairness is only concerned with persistent and non-redundant clauses, but it is not safe, because inference messages carrying non-persistent clauses may not be received in the form they were broadcast. 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 intermingled with the receive-and--forward operation. The latter 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. This is the case, for instance, if contraction is done by localized image sets. 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 atomic primitives.

The following theorem summarizes all the conditions for proof reconstruction:

Theorem 4.1 *Given a distributed theorem proving strategy $\mathcal{C}$ such that*

1. *$\mathcal{C}$ has a globally unambiguous naming scheme,*
2. *$\mathcal{C}$ deletes the clauses in $S \cup V$ by backward contraction and saves in D the clauses deleted by backward contraction,*
3. *$\mathcal{C}$ satisfies the three hypotheses of Theorem 3.1 for uniform fairness and*
4. *$\mathcal{C}$ has a comprehensive and safe communication scheme,*

then for all derivations
$T_0^k \vdash_{\mathcal{C}} T_1^k \vdash_{\mathcal{C}} \ldots T_i^k \vdash_{\mathcal{C}} \ldots$,
with $T = (S; V; CP; MI; MO; D)$, 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 $at(\Box)$ from its final state $(S; V; CP; MI; MO; D)^i_{h_i}$.

Proof: we show that all clauses in $at(\Box)$ can be found in $(S; V; CP; M_1; M_2; D)^i_{h_i}$. The proof is by induction on the depth m of $at(\Box)$.
Base: if $m = 1$, then $at(\Box) = (\Box, f(\psi_1, \ldots \psi_r))$, where $\psi_1, \ldots \psi_r$ are input clauses and f is the inference rule that p_i uses to generates $\Box$ from $\psi_1, \ldots \psi_r$ at stage h_i. Thus, $\psi_1, \ldots \psi_r$ are available at p_i at stage h_i.
Induction hypothesis: all clauses in $at(\Box)$ up to depth $m = q$ are retrievable by p_i.
Induction step: let φ be a clause at depth q in $at(\Box)$ 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 show that, given their identifiers, p_i can retrieve $\psi_1, \ldots \psi_r$ at stage h_i. 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 $\psi_1, \ldots \psi_r \in (S \cup V)^i_{h_i}$ and therefore can be retrieved at stage 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. Then, $\psi_j \in D^i_{h_i}$ and $\psi_1, \ldots \psi_r \in (S \cup V \cup D)^i_{h_i}$.
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 before termination. Since the communication scheme is also safe, they were received by all processes before termination. 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 3.1, $MI^i_\infty = MI^i_{h_j} = \emptyset$. Thus, $\psi_1, \ldots \psi_r$ are moved from the MI component to $S \cup V$. For all j, $1 \leq j \leq r$, either ψ_j is persistent or it is not. If it is persistent, then $\psi_j \in (S \cup V)^i_{h_j}$. If it is not persistent, then, since it is in $S \cup V$, it must have been deleted by backward contraction. Since the strategy saves in D the clauses deleted by backward contraction, we have $\psi_j \in D^i_{h_j}$. In both cases, ψ_j can be retrieved by p_i at stage h_i. $\Box$

Modified Clause-Diffusion has a globally unambiguous naming scheme (Section 3.2), because it has no repetitions and it is invoked for all non-trivial new clauses, including those generated by backward contraction. The communication scheme is comprehensive, because non-trivial new clauses are broadcast upon generation (Section 3.1). If the specific strategy is fair, has a safe communication scheme and saves clauses deleted by backward contraction, then proof reconstruction is guaranteed.

5 Discussion

We studied the problem of proof reconstruction in the context of distributed theorem proving by peer, concurrent, deductive processes with asynchronous communication and distributed memory. A distributed derivation succeeds as soon as one of the deductive processes does. The proof reconstruction problem is to guarantee that the successful process be able to reconstruct the distributed proof based solely on the final state of its data base. 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 failures of proof reconstruction that we observed in experiments with our Clause-Dif-

fusion prototypes, we focused on the components of a Clause-Diffusion strategy that are relevant to the reconstruction of proofs: the communication scheme, the naming scheme and the treatment of the new clauses, reduced forms of previously existing clauses, generated by backward contraction. Not surprisingly, backward contraction, the feature of contraction-based strategies that makes their parallelization difficult, turned out to be crucial for proof reconstruction also. Based on this analysis, we proposed a modified Clause-Diffusion method with new communication scheme, naming scheme and treatment of raw clauses introduced by backward contraction. We proved that modified Clause-Diffusion is fair and thus complete like original Clause-Diffusion. Then, we showed that modified Clause-Diffusion guarantees proof reconstruction and thus is a solution to our proof reconstruction problem.

We remark that 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 data base in shared memory and proof reconstruction can be done like in the sequential case. Similarly, proof reconstruction in a distributed system with peer processes is more challenging than 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 data base of the master. More generally, the more centralized is the control and the more predictable is the communication, the simpler is the book-keeping and thus proof reconstruction. For instance, in the Team-Work method of [1], the data bases of the deductive processes are periodically merged, so that proof reconstruction can also be done in a single data base. In this paper we 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.

The emphasis of this paper was not on the efficiency of distributed theorem proving strategies. We refer to previous work, e.g. [4, 5, 7], for considerations on the strengths and limitations of the Clause-Diffusion approach in this regard. Since the feasibility of proof reconstruction requires that all clauses used as premises are broadcast, one may conjecture that a strategy without the proof reconstruction property may need less communication and therefore be more efficient. On the other hand, uniform fairness, which is used to establish fairness and thus completeness of the strategy, requires that all persistent, non-redundant clauses are broadcast. Since in practice a strategy cannot predict which clauses will be persistent, we think that proof reconstruction does not pose much higher requirements than those required by completeness. This is written under the assumption that the strategy gives highest priority to contraction, and, therefore, does not use as premises clauses which are presently redundant. For experimental purposes and study of performances, one may implement both the original Clause-Diffusion without proof reconstruction and the modified Clause-Diffusion with proof reconstruction.

In addition to being a desirable property, we feel that the proof reconstruction issue led us to polish and streamline Clause-Diffusion significantly. The continuation of this research will be to realize a new Clause-Diffusion prototype which implements modified Clause-Diffusion.

Acknowledgements

I would like to thank Jieh Hsiang for his comments on an earlier version of this paper.

References

[1] J.Avenhaus and J.Denzinger, Distributing Equational Theorem Proving, in C.Kirchner (ed.), *Proceedings of the Fifth Conference on Rewriting Techniques and Applications*, Montréal, Canada, June 1993, Springer Verlag, Lecture Notes in Computer Science 690, 62–76, 1993.

[2] L.Bachmair and H.Ganzinger, Non-Clausal Resolution and Superposition with Selection and Redundancy Criteria, in A.Voronkov (ed.), *Proceedings of Logic Programming and Automated Reasoning*, Springer Verlag, Lecture Notes in Artificial Intelligence 624, 273–284, 1992.

[3] M.P.Bonacina and J.Hsiang, On fairness in distributed deduction, in P.Enjalbert, A.Finkel and K.W.Wagner (eds.), *Proceedings of the Tenth Symposium on Theoretical Aspects of Computer Science*, Würzburg, Germany, February 1993, Springer Verlag, Lecture Notes in Computer Science 665, 141–152, February 1993.

[4] M.P.Bonacina and J.Hsiang, Distributed Deduction by Clause-Diffusion: the Aquar-

ius Prover, in A.Miola (ed.), *Proceedings of the Third International Symposium on Design and Implementation of Symbolic Computation Systems*, Gmunden, Austria, September 1993, Springer Verlag, Lecture Notes in Computer Science 722, 272–287, September 1993.

[5] M.P.Bonacina and J.Hsiang, The Clause-Diffusion methodology for distributed deduction, to appear in D.A.Plaisted (ed.), *Fundamenta Informaticae*, Special Issue on Term Rewriting Systems.

[6] M.P.Bonacina and J.Hsiang, Parallelization of deduction strategies: an analytical study, to appear in the *Journal of Automated Reasoning*.

[7] M.P.Bonacina and W.W.McCune, Distributed theorem proving by *Peers*, in A.Bundy (ed.), *Proceedings of the Twelfth International Conference on Automated Deduction*, Nancy, France, June 1994, Springer Verlag, Lecture Notes in Computer Science 814, 841–845, 1994.

[8] R.Butler and E.L.Lusk, User's Guide to the p4 Programming System, Technical Report ANL-92/17, Mathematics and Computer Science Division, Argonne National Laboratory, October 1992.

[9] W.W.McCune, OTTER 2.0 Users Guide, Technical Report ANL-90/9, Mathematics and Computer Science Division, Argonne National Laboratory, March 1990.

[10] W.W.McCune, Otter 3.0 Reference Manual and Guide, Technical Report ANL-94/6, Mathematics and Computer Science Division, Argonne National Laboratory, January 1994.

[11] C.B.Suttner and J.Schumann, Parallel Automated Theorem Proving, in L.Kanal, V.Kumar, H.Kitano and C.B.Suttner (eds.), *Parallel Processing for Artificial Intelligence*, Elsevier, 1994.

[12] L.Wos, D.Carson and G.Robinson, Efficiency and completeness of the set of support strategy in theorem proving, *Journal of the ACM*, Vol. 12, 536–541, 1965.