On subsumption in distributed derivations

Maria Paola Bonacina
Department of Computer Science
University of Iowa
Iowa City, IA 52242-1419, USA

Jieh Hsiang
Department of Computer Science
National Taiwan University
Taipei, Taiwan

Abstract

In this paper we study the subsumption inference rule in the context of distributed deduction. It is well known that the unrestricted application of subsumption may destroy the fairness and thus the completeness of a deduction strategy. Solutions to this problem in sequential theorem proving are known. We observe that in distributed automated deduction, subsumption may also thwart *monotonicity*, a dual property of soundness, in addition to completeness. Not only do the solutions for the sequential case not apply, even proper subsumption may destroy monotonicity in the distributed case.

We present these problems and propose a general solution that treats subsumption as a composition of a replacement inference rule, *replacement subsumption*, and a deletion inference rule, *variant subsumption*. (Proper subsumption, in this case, becomes a derived inference rule.) We define a new *distributed subsumption* inference rule, which has all the desirable properties: it allows subsumption, including subsumption of variants, in a distributed derivation, while preserving fairness and monotonicity. It also works in both sequential and distributed environments.

We conclude the paper with some discussion of the different behavior of subsumption in different architectures.

Keywords: Automated theorem proving, distributed deduction, contraction, subsumption

1 Introduction

A widely used approach to curb the explosion of search space in automated deduction is the employment of inference rules that eliminate redundant data from the database. The earliest and the most well known is the *subsumption principle*, which discards a clause φ, if the current set of clauses contains another clause ψ such that $\forall x\psi \supset \forall y\varphi$ is valid [9, 11]. This principle is usually not implemented in all its generality. Practical inference rules based on the subsumption principle include *clausal subsumption* [9] and *functional subsumption* [6]. In clausal subsumption, clause ψ subsumes clause φ if $\psi\sigma \subseteq \varphi$ for some substitution σ and if ψ has no more literals than φ. We denote it as $\varphi \trianglerighteq \psi$, that is, φ is greater than ψ in the *subsumption ordering*. If both $\varphi \trianglerighteq \psi$ and $\psi \trianglerighteq \varphi$, then ψ and φ are *variants* of each other, written $\psi \stackrel{\bullet}{=} \varphi$. If $\varphi \trianglerighteq \psi$ and $\varphi \stackrel{\bullet}{\neq} \psi$, then ψ *properly subsumes* φ, written $\varphi \triangleright \psi$. In functional subsumption, $l \simeq r$ subsumes $p \simeq q$ if $p = c[l\sigma]$ and $q = c[r\sigma]$ for some context c and substitution σ. Similar to clauses, this condition can be used to define an ordering, the *encompassment ordering* on equations $\trianglerighteq$ [5]. If $(p \simeq q) \trianglerighteq (l \simeq r)$ and $(l \simeq r) \trianglerighteq (p \simeq q)$, then $p \simeq q$ and $l \simeq r$ are *variants*. If $(p \simeq q) \trianglerighteq (l \simeq r)$ and $(p \simeq q) \stackrel{\bullet}{\neq} (l \simeq r)$,

then $(p \simeq q) \blacktriangleright (l \simeq r)$. Note that both the proper subsumption and proper encompassment orderings are well-founded relations.

The classes of subsumption steps can also be categorized according to when the clauses are generated. A subsumption step where ψ subsumes φ is called a *forward subsumption* step, if ψ is generated earlier than φ in the derivation, otherwise it is a *backward subsumption* step.

It is well known from [9] that unrestricted application of subsumption, and especially *backward subsumption of variants*, may destroy the completeness of a strategy that is otherwise complete. The reason is that a derivation may generate an infinite sequence of variants of the same clause, each of which subsumes its predecessor and prevents any clause in the sequence from being selected for other inferences necessary for obtaining a proof. Technically, the source of the problem is that the subsumption ordering is not well founded. Because of this difficulty, many inference systems include only proper subsumption, since it is well founded and preserves completeness (e.g., [2, 12]). On the other hand, most existing first-order theorem provers (e.g., [10]) do feature subsumption, not just proper subsumption, because eliminating variants is very useful in practice. Completeness is maintained usually by combining the proper subsumption ordering with some other ordering to sort variants. For instance, the prover may associate to each clause its age for the purpose of subsumption and a clause is forbidden to subsume an older variant. This provision prevents the violation to completeness illustrated in [9].

The problem with subsumption becomes much more serious, however, in a distributed derivation. By distributed theorem proving we mean the possibility of having several deduction processes proceeding *concurrently* and *asynchronously*. Each one of them owns a portion of the database of clauses and they exchange clauses by *message passing*. A distributed derivation is made of the derivations developed by these processes. Clauses may be *duplicated*; that is, the same clause may appear in different derivations. In this context, two unprecedented difficulties arise. First, it may happen that a combination of concurrent steps of subsumption of variants deletes all the variants of a clause. This makes the distributed derivation *non-monotonic*, as theorems of the original theory may be lost. Thus, in addition to the possibility of losing completeness, subsumption of variants causes the even more fundamental problem of losing monotonicity. The solutions known for the sequential case cannot be extended straightforwardly to the distributed one. For instance, sorting variants by age is not sufficient, because the concurrent processes are asynchronous, so that there is no general way to establish that one clause has been generated earlier than another. Second, not only subsumption of variants but proper subsumption itself may also destroy monotonicity in a distributed derivation. This may happen if a process uses a received message ψ to subsume and delete a clause φ from its own database. Under certain message-passing schemes, there is no guarantee that ψ is stored in the database of another process. If it is not, both φ and ψ may be lost, thus violating monotonicity.

In this paper we analyze these issues and provide a comprehensive solution for both sequential and distributed subsumption. We show that the source of the problem lies in a misconception of the subsumption inference rule. We reason that proper subsumption is not deletion of a clause φ by a more general clause ψ, but rather *replacing* φ by ψ. This difference cannot be easily appreciated in the sequential case, where φ and ψ are stored in the same database. But it is significant in a distributed environment, where inferences may involve clauses belonging to remote databases. Based on this understanding of subsumption, we refine the subsumption inference rule into two rules: *replacement subsumption* and *variant subsumption*. (Under this framework sequential proper subsumption becomes a derived inference rule). In the distributed case,

variations of the two inference rules are combined into a *distributed subsumption* inference rule, which has all the desirable properties of subsumption. It allows proper subsumption between clauses stored in remote databases without violating monotonicity, and subsumption of variants while preserving monotonicity and completeness. For the latter we also present a way of achieving well-foundedness for variant subsumption without assuming the existence of a global clock.

The rest of the paper is organized as follows. First, we recall a few basic notions in theorem proving; we outline our approach to distributed deduction, called in [3] *distributed deduction by Clause-Diffusion*, and introduce a notion of distributed derivations. We then present in detail the problems with subsumption in sequential and distributed derivations. The new subsumption inference rules are then described, and we show they provide a solution to all the mentioned problems.

Acknowledgements

This work was done while both authors were with the Department of Computer Science of the State University of New York at Stony Brook. In addition to the support from the Department and the University, we acknowledge support from the National Science Foundation under grant CCR-8901322. The first author was also supported by a fellowship of Università degli Studi di Milano, Italy. The manuscript was submitted while the first author was visiting the Mathematics and Computer Science Division of Argonne National Laboratory.

2 Distributed deduction

2.1 Basic concepts in theorem proving

A *theorem proving problem* consists in deciding, given a set of clauses S and a clause φ, whether φ is a theorem of S, written $\varphi \in Th(S)$. A theorem proving strategy $\mathcal{C}$ is specified by a set of *inference rules* I and a *search plan* Σ. The inference rules can be further characterized into *expansion* inference rules and *contraction* inference rules. Expansion rules derive new clauses from existing ones and add them to the database. Resolution and paramodulation are two typical expansion inference rules. Contraction rules delete existing clauses and possibly replace them by logically equivalent ones, which are smaller according to some well-founded ordering. Subsumption, tautology elimination, simplification, and normalization are examples of contraction inference rules. While expansion rules are sufficient for completeness, contraction rules are often crucial for the practical success of theorem proving, because they contain the growth of the search space [1, 7, 13].

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

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

is constructed. A derivation is required to preserve the theory (i.e., for all $i \geq 0$, $Th(S_i) \subseteq Th(S_{i+1})$), without extending it, (i.e., for all $i \geq 0$, $Th(S_{i+1}) \subseteq Th(S_i)$). The latter is the basic property of *soundness*, while the former is usually termed *monotonicity*. A derivation is *successful* if it reaches a solution. A theorem-proving strategy $\mathcal{C}$ is *complete* if, whenever the input target is indeed a theorem, the derivation

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

2.2 Distributed theorem proving

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

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

The separation of residents and inference messages can be used to partition the search space at the inference level as well. Using the paramodulation inference rule as an example, one may establish that the inference messages are paramodulated *into* the residents, but not vice versa. This restriction has two purposes. First, it distributes the expansion inference steps among the nodes. Second, it prevents a systematic duplication of steps: if this rule were not in place, the paramodulation steps between two residents ψ_1 of p_1 and ψ_2 of of p_2 would be performed twice, once when ψ_1 visits p_2 and once when ψ_2 visits p_1. Other expansion inference rules can be treated in a similar way, including non-binary inference rules [3]. While subdividing the expansion steps serves its purpose, it is not productive to subdivide the contraction steps, since the motivation behind contraction is to keep the database always at the minimal. In a *contraction-based strategy*, an expansion step should be performed only if all the premises are fully reduced, at least with respect to the local database. To ensure this, we require that each processor keep both its residents and received inference messages fully contracted.

Let us call the clause newly generated from an expansion step a *raw clause*. In the presence of contraction rules, a raw clause should not become a resident until it has been maximally reduced. Thus, our

method also features a number of *distributed contraction schemes* [3] to reduce a clause with respect to the global, distributed database. After contraction, a raw clause becomes a *new settler*, i.e. a clause to be given to the allocation algorithm to be assigned to some node.

This is the basic working of the "Clause-Diffusion" approach to distributed automated deduction: inter-contraction and local expansion inferences at the nodes among residents and inference messages, distributed contraction, allocation of new settlers, and mechanisms for passing inference messages. By specifying the inference mechanism I, the search plan Σ to schedule inference steps and communication steps, the allocation algorithm, the distributed contraction scheme, and the algorithms for routing and broadcasting of messages, one obtains a specific strategy. We refer to [3] for full detail of the methodology and its implementation.

The above elements are summarized in the following notion of *distributed derivation*: every processor p_k computes a derivation

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

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

A distributed derivation is the collection of the asynchronous derivations computed by the nodes, and it succeeds as soon as the derivation at one node finds a proof. The *state* of the derivations at processor p_k and stage i is represented by the tuple $(S; M; CP; NS)_i^k$. More components may be added, if indicated by a specific strategy. A step in a distributed derivation can be either an *inference* step or a *communication* step. For instance, sending an inference message for $\psi \in S^k$ from node p_k to an adjacent node p_j can be written as $(S^k \cup \{\psi\}, M^j) \vdash (S^k \cup \{\psi\}, M^j \cup \{\psi\})$.

3 Subsumption in distributed derivations

3.1 The problem with subsumption and fairness

A strategy that allows unrestricted application of subsumption is not complete in general. The following example, appeared originally in [8] and reported in [9] (pages 207-208), illustrates this phenomenon. Consider a resolution theorem proving strategy whose search plan uses the *set of support* restriction and sorts the set of support as a first-in-first-out queue. Since resolution is refutationally complete, and the first-in-first-out set of support search plan is fair, it follows that the strategy is complete. It is no longer complete, however, if subsumption is added to the inference system and the search plan applies it as soon as possible. Let the input set of clauses be (1) $P(x, a)$, (2) $P(f(x), y) \vee \neg P(x, y)$, (3) $\neg Q(y) \vee \neg P(x, y)$ and (4) $Q(a)$, where (1) $P(x, a)$ is originally in the set of support. The search plan selects clause (1) and resolves it first with (2) to generate (5) $P(f(x), a)$ and then with (3) to yield (6) $\neg Q(a)$. The two new clauses are added in this order to the set of support. Next, the search plan selects (5) and resolves it with (2) and with (3), generating (7) $P(f(f(x)), a)$ and (8) $\neg Q(a)$, respectively. Clause (8) back-subsumes (6), so that the search plan selects (7) next and generates (9) $P((f(f(fx))), a)$ and (10) $\neg Q(a)$. Again, (10) back-subsumes (8) and (9) is selected. The derivation proceeds indefinitely, always missing to use $\neg Q(a)$ to resolve with $Q(a)$ and obtain the empty clause.

As remarked in [9], the root of the problem is that the subsumption ordering is *not* well founded. In the above example, the derivation generates an infinite sequence $\neg Q(a) \succeq \neg Q(a) \succeq \ldots$, where each element subsumes its predecessor, so that the resolution of $\neg Q(a)$ and $Q(a)$ is never selected. Thus, a search plan which applies eagerly a non-well-founded contraction rule is not fair, and the strategy is not complete. The usual solution is to label the clauses in some well-founded ordering and perform subsumption of variants according to the ordering. For instance, the restriction of *forward subsumption before backward subsumption* [9] can be interpreted as labeling the clauses according to the time they are generated.

3.2 The problem with subsumption and monotonicity

In distributed deduction, unrestricted application of subsumption may violate not only fairness, but also monotonicity of the derivation. Intuitively, the reason is that in a distributed derivation it is possible to have *duplicated data*. That is, the same clause φ may be present at the same time in the derivation both as resident at one node and as inference message at another. First, we see how subsumption of variants may be harmful. Envision two variants φ and ψ residing at nodes p_i and p_j, respectively. Assume also that p_i and p_j have received, respectively, a copy of ψ and φ as inference messages. There are several possible scenarios on how subsumption may be done, but none of them is satisfactory. If the strategy establishes that the messages be used to subsume residents first, then *both* residents will be deleted, destroying monotonicity. If residents subsume inference messages first, then both messages will be subsumed, while both residents will remain intact. This defeats the purpose of subsumption since both variants are still in the global database.

More surprising, even proper subsumption may destroy monotonicity: consider a distributed strategy where messages are subject to contraction along their routes, and messages are deleted at each node after necessary inferences are performed. The first assumption applies to almost all strategies since it reduces the number of redundant messages. The second assumption may be necessary if the nodes do not have sufficient local memory. Let $P(f(x))$ and $P(g(y))$ be residents at nodes p_i and p_j respectively. Assume that the message carrying $P(f(x))$ from p_i has been reduced by an equation $f(u) \simeq u$ at an intermediate node and arrives at p_j as $P(x)$, and that the message carrying $P(g(y))$ from p_j has been reduced by an equation $g(v) \simeq v$ at an intermediate node and arrives at p_i as $P(y)$. If proper subsumption is applied at the two nodes, then both residents $P(f(x))$ and $P(g(y))$ will be subsumed and deleted, resulting in the loss of monotonicity:

P_i	P_j	time ↓
resident P(f(x))	resident P(g(y))	initial state
resident P(f(x)) message P(y)	resident P(g(y)) message P(x)	messages arriving
message P(y)	message P(x)	after subsumption
empty	empty	messages consumed

3.3 Sequential subsumption revisited

A satisfactory solution to the above problems requires a re-examination of the concept of subsumption itself. The essence of contraction is basically replacing a clause by an equivalent or more general one in the database. Thus, *tautological deletion* is replacing a tautology by the clause *true*, which is assumed to be present in any database of clauses. Similarly, in proper subsumption a clause ψ is deleted if there is a clause φ in the bata base that is more general than ψ ($\psi \succ \varphi$). What is important to observe here is that in the latter, ψ is *replaced* by φ, not by the clause *true*. In other words, a proper subsumption step is in fact composed of two steps: first replacing ψ by a variant $\varphi\prime$ of φ if $\psi \succ \varphi$, then deleting $\varphi\prime$ since its variant φ is already in the database. In the sequential case, the conventional view of subsumption is sufficient since the inference process assumes one database. In distributed deduction, however, the separation of the two steps becomes crucial since φ, which subsumes ψ, may not belong to the local database on which a deduction process is operating. An outright deletion of ψ at one node may result in the loss of information in the global database, and the consequent loss of monotonicity.

With this understanding, we refine subsumption into two inference rules, the *replacement of a clause by a more general one* and the *subsumption of variants*:

Replacement Subsumption (R-Subsumption)

$$\frac{A \cup \{\varphi_1, \varphi_2\}}{A \cup \{\varphi_1, \varphi_1'\}} \quad \varphi_2 \succ \varphi_1,\ \varphi_1' \stackrel{\bullet}{=} \varphi_1,$$

Variant Subsumption (V-Subsumption)

$$\frac{A \cup \{\varphi_1, \varphi_2\}}{A \cup \{\varphi_1\}} \quad \varphi_2 \stackrel{\bullet}{=} \varphi_1.$$

In R-subsumption, a clause is replaced by a variant of the clause that subsumes it, and in V-subsumption a variant is deleted. *Proper subsumption* becomes a derived inference rule, equivalent to the composition of an R-subsumption step followed by a V-subsumption step:

Proper Subsumption (P-Subsumption)

$$\frac{\dfrac{A \cup \{\varphi_1, \varphi_2\}}{A \cup \{\varphi_1, \varphi_1'\}} \quad \varphi_2 \succ \varphi_1,\ \varphi_1' \stackrel{\bullet}{=} \varphi_1}{A \cup \{\varphi_1\}} \quad \varphi_1' \stackrel{\bullet}{=} \varphi_1$$

or, in short:

$$\frac{A \cup \{\varphi_1, \varphi_2\}}{A \cup \{\varphi_1\}} \quad \varphi_2 \succ \varphi_1.$$

The conventional approach of sorting variants by age to decide which should be deleted can also fit in this framework by refining variant subsumption into

Ordered Variant Subsumption

$$\frac{A \cup \{\varphi_1, \varphi_2\}}{A \cup \{\varphi_1\}} \quad \varphi_2 \stackrel{\bullet}{=} \varphi_1, t_2 > t_1,$$

where t_1 and t_2 are the birth-times of φ_1 and φ_2 respectively. One can replace V-subsumption by Ordered V-subsumption and still obtain the same proper subsumption inference rule, because the variant φ_1' resulting from the R-subsumption step conceivably has a greater birth-time than φ_1, which is already in the database.

3.4 The inference rules for distributed subsumption

In this section we present distributed versions of the subsumption inference rules. As in the sequential case, distributed proper subsumption is derived as the composition of distributed replacement and variant subsumption rules. If the derivation is sequential, the distributed inference rules reduce to their sequential (ordered) counterparts.

3.4.1 Distributed variant-subsumption

The basic idea is to attach to the clauses some extra information so that well-foundedness can be established, even in the absence of a global clock. Assume that each node has its own clock. We call the local time when a clause φ is produced by an expansion inference step the *inference-time* of φ. When φ settles down as a resident at p_i, it is assigned the current time of p_i as its *birth-time*. It is reasonable to assume that no two resident clauses at the same node receive the same birth-time and that no two clauses have the same inference-time at the same node. When an inference message carrying the information about a specific resident is sent to other nodes, it also includes the birth-time of the resident as a *time-stamp*. The format for an inference message is $< \varphi, p_i, x >$, where p_i is the sender and x is the birth-time of φ at p_i. In addition, the *reception-time* of an inference message, received at a node p_j, is defined as the time at p_j's clock, when the message is received at p_j. Any clause that has not yet settled as a resident, such as a raw clause newly generated by an expansion inference, is assumed to have birth-time ∞.

We can now associate with every clause a measure, that depends on the attributes of the clause and on the node where the clause is being considered.

Definition 3.1 *The measure ds ("*distributed subsumption*") is defined as follows:*

- *for a resident φ at p_k, with birth-time t, $ds(\varphi, k) = (t, k, \_)$,*
- *for an inference message $< \varphi, p_i, x >$ received at node p_k, $ds(\varphi, k) = (x, i, y)$, where y is its reception-time at p_k, and*
- *for a raw clause φ at p_k, born at p_k with inference-time s, $ds(\varphi, k) = (\infty, s, \_)$.*

Next, we define an ordering to compare clauses based on this measure.

Definition 3.2 *Let $>$ denote the ordering on natural numbers extended with ∞ as the maximal element. Given two clauses φ and ψ in $S^k \cup M^k \cup CP^k$ at some node p_k, the ordering $\succ_{ds}$ is defined as follows: $\varphi \succ_{ds} \psi$ if and only if $ds(\varphi, k) \succ_{dm} ds(\psi, k)$, where $\succ_{dm}$ is the threefold lexicographic combination of the ordering $>$.*

Then, we define distributed variant-subsumption as follows.

Distributed Variant Subsumption (DV-subsumption):

$$\frac{A \cup \{\varphi_1, \varphi_2\}}{A \cup \{\varphi_1\}} \quad \varphi_2 \stackrel{\bullet}{=} \varphi_1,\ \varphi_2 \succ_{ds} \varphi_1,$$

where A is a set of clauses such as the union $S^k \cup M^k \cup CP^k$ at a node p_k.

3.4.2 Distributed variant-subsumption: monotonicity and fairness preserved

We consider two variants φ and ψ. If the two clauses are two residents at the same node p_k, then the birth-time decides, and the younger resident (the one with larger birth-time) is subsumed. In this way distributed subsumption naturally incorporates the criterion of sorting variants by age used in sequential derivations. If one of the clauses is a raw clause and the other one is either a resident or an inference message, then the raw clause is subsumed, since $\infty > t$ for all t. If both clauses are raw clauses, they have the same birth-time ∞ and the ordering on the inference-times is used to break the tie. Since raw clauses are local to each node, two raw clauses are guaranteed to have different inference times.

If the two clauses are both inference messages, we compare the time-stamps, so that the treatment of inference messages is consistent with the treatment of residents. But two inference messages may have the same time-stamps, if they are taken from clocks of different nodes. When this happens, we decide based on the second component of the measure ds (i.e., the indices of the nodes). Under certain broadcasting schemes, a node may receive inference messages $m_1 =< \psi, p_k, x >$ and $m_2 =< \psi', p_k, x >$, originated from the same node p_k and with the same time-stamp. The two messages originally carried the same clause φ, which has been reduced by contraction to ψ in m_1 and to ψ' in m_2. In this case, the reception-times are compared, and the message received first subsumes the other one.

The last and most interesting case is when one of the variants is an inference message and the other one a resident. We prove that distributed subsumption *deletes exactly one of the two variants*:

Theorem 3.1 *Let variants φ and ψ be two residents at p_i and p_j respectively. Under distributed subsumption, exactly one of them will be subsumed.*

Proof: Let φ be a resident at p_i with birth-time t_1 and ψ be a resident at p_j with birth-time t_2, such that φ and ψ are variants. If $i = j$ (i.e., the two clauses are residents at the same node), we have $ds(\varphi, i) = (t_1, i, \_)$ and $ds(\psi, i) = (t_2, i, \_)$. Since t_1 and t_2 are taken at the same clock, it is $t_1 \neq t_2$. Therefore, φ subsumes ψ if $t_1 < t_2$ and ψ subsumes φ if $t_2 < t_1$. If $i \neq j$, let $m_1 =< \varphi, p_i, t_1 >$ and $m_2 =< \psi, p_j, t_2 >$ be two inference messages from the two residents: m_1 is received at p_j at time y_1 of p_j's clock, while m_2 is received at p_i at time y_2 of p_i's clock. At p_i, we have $ds(\varphi, i) = (t_1, i, \_)$ and $ds(\psi, i) = (t_2, j, y_2)$. At p_j, we have $ds(\varphi, j) = (t_1, i, y_1)$ and $ds(\psi, j) = (t_2, j, \_)$. If $t_2 < t_1$, then $< \psi, p_j, t_2 >$ subsumes φ at p_i and ψ subsumes $< \varphi, p_i, t_1 >$ at p_j. If $t_1 < t_2$, then $< \varphi, p_i, t_1 >$ subsumes ψ at p_j and φ subsumes $< \psi, p_j, t_2 >$ at p_i. In both cases, one and only one of the two residents is deleted. If $t_1 = t_2$, the indices i and j are compared. Since $i \neq j$, we have either $i < j$ or $j < i$, so that again one and only one of the two residents is deleted. □

This proves that distributed subsumption does not harm monotonicity by deleting all variants of a clause. Since the ordering $\succ_{ds}$ is well founded, the violation of fairness showed in Section 3.1 will not happen either.

3.4.3 Distributed proper subsumption

Similar to the sequential case, distributed replacement subsumption replaces a clause by a variant of a more general clause *at a local node*. Furthermore, since the data involved include more than just the clauses themselves, other information such as birth-time and inference-time also needs to be updated.

Distributed Replacement Subsumption (DR-Subsumption)

$$\frac{A \cup \{\varphi_1, \varphi_2\}}{A \cup \{\varphi_1, \varphi_1'\}} \quad \varphi_2 \succcurlyeq \varphi_1,\ \varphi_1' \stackrel{\bullet}{=} \varphi_1.$$

Let z be the time of p_k's clock when the step is performed.

- If φ_2 is a resident with birth-time t, $< \varphi_2, t > \in S_k$, it is replaced by $< \varphi_1', z > \in S_k$ with $ds(\varphi_1', k) = (z, k, \_)$. Furthermore, if φ_1 is an inference message $< \varphi_1, p_i, x > \in M^k$, then its time-stamp is reset to ∞, that is, we update it into $< \varphi_1, p_i, \infty > \in M^k$.
- If φ_2 is an inference message $< \varphi_2, p_i, x > \in M^k$, then it is replaced by $< \varphi_1', p_i, z > \in M^k$. The reception-time of $< \varphi_1', p_i, z >$ at p_k is also updated to z, so that $ds(\varphi_1', k) = (z, k, z)$.
- If φ_2 is a raw clause with inference-time s, $< \varphi_2, s > \in CP_k$, it is replaced by $< \varphi_1', z > \in CP_k$ with $ds(\varphi_1', k) = (\infty, k, \_)$.

Distributed proper subsumption is the composition of DR-subsumption and DV-subsumption:

Distributed Proper Subsumption (DP-Subsumption)

$$\frac{\dfrac{A \cup \{\varphi_1, \varphi_2\}}{A \cup \{\varphi_1, \varphi_1'\}} \quad \varphi_2 \succcurlyeq \varphi_1,\ \varphi_1' \stackrel{\bullet}{=} \varphi_1}{A \cup \{\varphi_1''\}} \quad \varphi_1'' \textit{ is } \varphi_1 \textit{ if } \varphi_1' \succ_{ds} \varphi_1,\ \varphi_1'' \textit{ is } \varphi_1' \textit{ if } \varphi_1 \succ_{ds} \varphi_1'.$$

The distributed proper subsumption rule needs to fulfill two requirements:

1. Noting that the DR-rule may create copies of the same variant (e.g. φ_1 and φ_1'), the DP-rule must ensure that either $\varphi_1' \succ_{ds} \varphi_1$ or $\varphi_1 \succ_{ds} \varphi_1'$ is true. This will not only secure fairness, but also guarantee maximal removal of redundancy.

2. If φ_2 in the above inference step is a resident at p_k, then the resulting clause φ_1'' must also be a resident at p_k. This results in the preservation of monotonicity.

To demonstrate that DP-subsumption indeed satisfies these conditions, we analyze each case based on the type of the clauses:

- φ_2 was a resident and thus φ_1' is a resident $< \varphi_1', z > \in S_k$:
  - if φ_1 is also a resident at p_k, φ_1' is deleted, because its birth-time z is more recent than the birth-time of φ_1; (thus, φ_1'' is φ_1, which is already a resident)
  - if φ_1 is an inference message received at p_k, φ_1 is deleted, because its time-stamp has been reset by the DR-subsumption step to ∞; (thus, φ_1'' is $\varphi_1\prime$)
  - if φ_1 is a raw clause, φ_1 is deleted, because its birth-time is ∞, which is greater than z (thus, φ_1'' is $\varphi_1\prime$).
- φ_2 was an inference message $< \varphi_2, p_i, x > \in M^k$, and thus φ_1' is an inference message $< \varphi_1', p_i, z > \in M^k$ with reception-time also equal to z:
  - if φ_1 is a resident at p_k, φ_1' is deleted, because its time-stamp z is more recent than the birth-time of φ_1;
  - if φ_1 is another inference message received at p_k, their measures are compared and one of them will be deleted, as explained above, when discussing DV-subsumption;
  - if φ_1 is a raw clause, φ_1 is deleted, because its birth-time is ∞, which is greater than z.
- φ_2 was a raw clause, and thus φ_1' is a raw clause with birth-time ∞ and inference-time z:
  - if φ_1 is a resident at p_k or an inference message received at p_k, φ_1' is deleted, because its birth-time is ∞;
  - if φ_1 is another raw clause, φ_1' is deleted, because its inference-time z is more recent than the inference-time of φ_1.

It follows that the definition of DP-subsumption can be rewritten as follows:

Distributed Proper Subsumption (DP-Subsumption)

$$\frac{A \cup \{\varphi_1, \varphi_2\}}{A \cup \{\varphi_1'\}} \quad \varphi_2 \succcurlyeq \varphi_1,\ \varphi_1' \stackrel{\bullet}{=} \varphi_1, \quad if\ \varphi_2 \in S^k,\ then\ \varphi_1' \in S^k.$$

Finally, we define *Distributed subsumption*:

Definition 3.3 *Given two clauses (or equations) φ and ψ in $S^k \cup M^k \cup CP^k$ at some node p_k,* D-subsumption *is DV-subsumption ,if $\varphi \stackrel{\bullet}{=} \psi$, DP-subsumption, if $\varphi \succcurlyeq \psi$ (or $\varphi \blacktriangleright \psi$).*

3.4.4 Distributed proper subsumption: monotonicity preserved

We consider now the problem with monotonicity resulting from proper subsumption of residents by inference messages (see Section 3.2). When an incoming inference message properly subsumes a resident, DP-subsumption prescribes replacing the resident by a variant of the message and deleting the message. Therefore, no resident is lost, but rather is replaced by a more general clause. For instance, in the example given in Section 3.2, the sequence of events becomes the following:

P_i	P_j	time ↓
resident P(f(x))	resident P(g(y))	initial state
resident P(f(x)) message P(y)	resident P(g(y)) message P(x)	messages arriving
resident P(y')	resident P(x')	after distributed proper subsumption
... ...	... ...	later ...
resident P(y') message P(x')	resident P(x') message P(y')	messages arriving
resident P(y')	empty	after distributed variant subsumption

The distributed scenario shows very effectively why viewing proper subsumption as replacement is preferable to viewing subsumption as deletion. Operationally, node p_i should not simply delete $P(f(x))$ upon receipt of $P(y)$, because p_i has not sufficient knowledge of the derivations at other nodes to establish that $P(y)$ is stored somewhere else. Conceptually, node p_i should not delete $P(f(x))$, because $P(f(x))$ is not trivially true. Rather, node p_i replaces $P(f(x))$ by $P(y')$, because $P(y)$ is more general than $P(f(x))$. Replacement-subsumption captures this concept. Also, our view of a proper subsumption step as the composition of a replacement-subsumption step and a variant-subsumption step appears very natural in the distributed context. Indeed, in the above example, global proper subsumption is achieved through two stages: at a first stage, $P(f(x))$ and $P(g(y))$ are replaced by $P(y')$ and $P(x')$, then, one of $P(x')$ and $P(y')$ is deleted. In the sequential case, it is not possible to appreciate the difference between proper subsumption as atomic deletion and proper subsumption as composition of replacement and deletion of a variant. In the distributed case, the difference is striking. Proper subsumption as atomic deletion violates monotonicity, whereas proper subsumption as composition does not, and it mimics effectively the interaction of remote residents through messages.

4 Discussion

The main obstacle with treating subsumption in sequential theorem proving is that subsumption of variants may destroy desirable properties such as fairness (and consequently completeness). In this paper we showed that in distributed deduction, subsumption can also destroy monotonicity, a dual property of soundness, and we provided a solution to these problems.

In a distributed environment, the additional troubles come from exchanging information through messages. We showed that in such a context, even proper subsumption may harm the monotonicity of a distributed derivation. Observation of this surprising phenomenon led us to rethink the meaning of the subsumption inference rule. We reasoned that subsumption is replacement of a clause by a more general one, rather than mere deletion of the less general clause. We distinguished between *replacement-subsumption* and *variant-subsumption*, and derived proper subsumption as their composition. We extended this three-rules framework to the distributed case and defined an inference rule of *distributed subsumption* which embeds distributed variant-subsumption and distributed proper subsumption as subcases. We proved that distributed subsumption has all the desirable properties: it prevents both the violations to monotonicity and the problems with subsumption between variants. Distributed subsumption has been implemented in our distributed prover *Aquarius* [3, 4].

Depending on the approaches and architectures, the problems of subsumption may arise with different gravity. In the following we briefly outline alternative scenarios.

The violations of monotonicity by proper subsumption may happen in any context where contraction steps and communication steps are interleaved. Specifically, an inference message φ originated at node p_i, is reduced to ψ at an intermediate node p_k and then ψ subsumes a resident ψ' at node p_j. It is not necessarily the case that $k \neq j$, that is, the simplification step may be applied at the receiver before the subsumption step. Thus, the problem with proper subsumption arises even if the message-passing scheme guarantees that a message issued by a node reaches all the other nodes in one hop, with no intermediate nodes. Our treatment covers this special case also.

In [3], we have considered also variants of the Clause-Diffusion methodology, where each node saves the inference messages it receives in a so-called *localized image set*. Under such provision, subsumption through messages, including subsumption of variants, does not destroy monotonicity. However, saving received messages is not per se a solution to the subsumption problem. When a message ψ from p_i is stored in the localized image set at node p_k, ψ is listed as a clause belonging to p_i. In the Clause-Diffusion methodology, preserving the ownership of the clauses is important to partition effectively the search space: p_k is not responsible for inference steps on ψ which pertain to its owner p_i. If all the variants of ψ that are stored as residents are deleted by uncontrolled subsumption, while ψ is saved as a message, monotonicity is not violated, but fairness may be violated: the message ψ will not be subject to all the inference steps, in particular expansion steps, which would be performed on ψ as a resident. Since these steps may be necessary to find a proof, the fairness of the strategy is thwarted. Distributed subumption provides a clean solution at the inference level, which is independent of specific assumptions about the treatement of messages.

Finally, in a purely shared memory approach, there is just one database, since all the clauses are held in the shared memory. Each deduction process has direct access to the entire database. In this respect, parallel deduction in shared memory is much closer to sequential deduction than distributed deduction. Indeed, the problem with proper subsumption as deletion may not arise in shared memory, because of the absence of messages. Even the problem with subsumption of variants may not occur, because shared memory may force the sequentialization of certain steps that are concurrent in a distributed environment. The following example illustrates this point. Let ψ and φ be variants. Then the step *"ψ subsumes φ"* needs write-access to φ and read-access to ψ, while the step *"φ subsumes ψ"* needs write-access to ψ and read-access to φ. The two steps are in read-write conflict on both premises. If the clauses are stored in a shared

memory that does not allow two concurrent processes to have read-access and write-access respectively to a same clause, the two steps cannot be executed concurrently. Although the shared-memory approach may avoid the monotonicity problems encountered in its distributed counterpart, this is mainly due to the sequentialization of contraction steps imposed by the restrictions of shared memory. Since contraction inferences are usually performed in abundance, sequentializing them may cause a serious bottleneck. In a distributed environment, on the other hand, such a bottleneck does not occur, and one can realize a larger degree of concurrency.

In this paper we focused on how to apply subsumption in distributed derivations *correctly*, that is, preserving fairness, thus completeness, and monotonicity. Once correctness is ensured, the question is the *effectiveness* of distributed subsumption in containing the generation and diffusion of subsumable clauses. Our first basic choice in approaching this problem is to use *contraction-based strategies*, that is strategies where contraction rules have higher priority than expansion rules. For instance, a subsumable clause will be subsumed before it can be used as a parent in resolution. However, in distributed deduction the priority of contraction steps can be enforced strictly only within the local database of each node: a node may use a clause for resolution without knowing that it can be subsumed by a clause at another node. Therefore, we also give higher priority to communication steps than expansion steps, so that the subsumer from the other node may arrive early enough to prevent the unwanted expansion step. While giving high priority to communication steps generally helps, it does not guarantee that subsumption will occur before resolution: inference steps are typically faster than communication steps, and, more important, the asynchronous nature of our approach to distributed deduction makes it impossible to guarantee a specific order of the steps.

The effective application of distributed subsumption is part of the general issue of *distributed global contraction* [3], that is, contraction (e.g., subsumption or normalization) with respect to a global distributed database. Distributed global contraction is critical in distributed deduction, because concurrent deductive processes have obviously the power of generating many redundant clauses (e.g., duplicates and subsumable clauses). In [3], we proposed several schemes for distributed global contraction, mechanisms that enable each node to use subsumers and simplifiers resident at other nodes. The above-mentioned provision of saving received inference messages in "localized image sets" is part of some of these schemes: on top of its residents, each node may use the saved inference messages as "global" simplifiers. This raises in turn other problems, such as how to keep these global simplifiers normalized. We refer to [3, 4] for a full account of these schemes and their implementation. The efficient realization of distributed global contraction, however, is still largely a challenge for current and future work.

References

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

[2] L.Bachmair and H.Ganzinger, Completion of First-Order Clauses with Equality by Strict Superposition, in M.Okada and S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Term Rewriting Systems*, Montréal, Canada, June 1990, Springer Verlag, Lecture Notes in Computer Science 516, 162–180, 1991.

[3] M.P.Bonacina, Distributed Automated Deduction, Ph.D. Thesis, Department of Computer Science, State University of New York at Stony Brook, December 1992.

[4] M.P.Bonacina and J.Hsiang, Distributed Deduction by Clause-Diffusion: the Aquarius 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, 1993.

[5] N.Dershowitz and J.-P.Jouannaud, Rewrite Systems, Chapter 15, Volume B, *Handbook of Theoretical Computer Science*, North-Holland, 1989.

[6] J.Hsiang and M.Rusinowitch, On word problems in equational theories, in Th.Ottman (ed.), *Proceedings of the Fourteenth International Conference on Automata, Languages and Programming*, Karlsruhe, Germany, July 1987, Springer Verlag, Lecture Notes in Computer Science 267, 54–71, 1987.

[7] D.Kapur and H.Zhang, RRL: a Rewrite Rule Laboratory, in E.Lusk, R.Overbeek (eds.), *Proceedings of the Ninth International Conference on Automated Deduction*, Argonne, Illinois, May 1988, Springer Verlag, Lecture Notes in Computer Science 310, 768–770, 1988.

[8] R.Kowalski, Studies in the completeness and efficiency of theorem proving by resolution, Ph.D. Thesis, University of Edinburgh, 1970.

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

[10] W.W.McCune, OTTER 2.0 Users Guide, Technical Report ANL-90/9, Argonne National Laboratory, Argonne, Illinois, March 1990.

[11] J.A.Robinson, A machine-oriented logic based on the resolution principle, *Journal of ACM*, Vol. 12, 23–41, 1965.

[12] M.Rusinowitch, Theorem-proving with Resolution and Superposition, *Journal of Symbolic Computation*, Vol. 11, N. 1 & 2, 21–50, January/February 1991.

[13] L.Wos, R.Overbeek and E.Lusk, Subsumption, a sometimes undervalued procedure, in J.-L.Lassez and G.Plotkin (eds.) *Computational Logic*, 3–41, MIT Press, 1991.