On the representation of parallel search in theorem proving

Maria Paola Bonacina[*]
Department of Computer Science – The University of Iowa

Abstract

This extended abstract summarizes two contributions from ongoing work on parallel search in theorem proving. First, we give a framework of definitions for parallel theorem proving, including inference system, *communication operators*, *parallel search plan*, *subdivision function*, *parallel strategy*, *parallel derivation*, *fairness* and *propagation of redundancy* for parallel derivations. A notion of a parallel strategy being a parallelization of a sequential strategy, and a theorem establishing a general relation between sequential fairness and parallel fairness are also given. Second, we extend our approach to the modelling of search to parallel search, covering inferences (expansion and contraction), behaviour of the search plan, subdivision of the search space and communication among the processes. This model allows us to study the behavior of many search processes on a single *marked search graph.* In the full paper, we plan to extend our methodology for the measure of the complexity of search in infinite spaces to parallel search, and apply it to obtain results in the comparison of strategies.

Introduction

Various approaches to parallel/distributed theorem proving have been proposed (e.g., [4, 10]). In order to evaluate how promising they may be, many of these methods have been implemented, and experiments have been reported. Empirical evaluation based on experiments, however, evaluates the implementations, rather than the theorem-proving strategies, so that it would be desirable to complement it with a machine-independent evaluation of the strategies. A main obstacle for this is represented by the infinite search spaces of theorem-proving problems, and the lack of mathematical tools to analyze strategies applied to infinite problems. In recent work [6], we began exploring an approach to the modelling of search and the measurement of the complexity of search in theorem proving. This extended abstract reports preliminary results on the extension of this approach to parallel theorem proving.

We are interested primarily in *forward-reasoning, contraction-based strategies*, such as those originated from rewriting-based methods on one hand, and the resolution-paramodulation paradigm on the other. In previous work [4], we analyzed how *coarse-grain parallelism*, that is, *parallel search*, seems to be most suitable for these strategies. Parallel search means that multiple deductive processes search in parallel the space of the problem. Approaches to parallel search differ in how they differentiate and compose the activities of the processes. Some *subdivide the search space* among them (e.g., *Clause-Diffusion* [5, 3]), others assign them *different search plans* (e.g., *Team-Work* [9] and [7]), and these two can also be combined. In all approaches, communication is needed to preserve completeness

[*]Department of Computer Science, The University of Iowa, Iowa City, IA 52242-1419, USA, bonacina@cs.uiowa.edu. Supported in part by the NSF with grants CCR-94-08667 and CCR-97-01508.

and combine the results of the searches. We consider parallel search with subdivision, although most notions can be generalized to using different search plans.

In the following, first we define precisely what are a *parallel search plan*, a *parallel strategy* and a *parallel derivation*. Since most previous works in parallel deduction were concerned with the description and implementation of specific methods, these definitions seem largely new. Then, we extend the model of [6] with *communication* and the *subdivision of the search space*. A key feature of [6] is the modelling of contraction. Similar to contraction, communication and subdivision make the search space *dynamic*. Thus, this extension reinforces the efficacy of our approach for modelling dynamic search spaces.

Parallel strategy: inference, communication, parallel search plan

The first component of a *theorem-proving strategy* is its *inference system*. Following [6], we characterize an inference rule as a function f^n, which takes an n-tuple of premises and returns a set of clauses to be added and a set of clauses to be deleted: $f^n\colon \mathcal{L}_\Theta^n \rightarrow \mathcal{P}(\mathcal{L}_\Theta) \times \mathcal{P}(\mathcal{L}_\Theta)$, where Θ is a signature, $\mathcal{L}_\Theta$ the language of clauses on Θ, and $\mathcal{P}(\mathcal{L}_\Theta)$ its powerset. An inference rule f^n is an *expansion inference rule* (e.g., resolution) if it generates new clauses with no deletions[1]: for all $\bar{x} \in \mathcal{L}_\Theta^n$, $\pi_2(f^n(\bar{x})) = \emptyset$. Given a well-founded ordering on clauses $(\mathcal{L}_\Theta, \succ)$, an inference rule f^n is a *contraction inference rule* (e.g., simplification by equations) if it delete clauses and possibly replace them by smaller ones: for some $\bar{x}$, $\pi_2(f^n(\bar{x})) \neq \emptyset$, and whenever $\pi_1(f^n(\bar{x})) \neq \emptyset$, for all $\varphi \in \pi_1(f^n(\bar{x}))$ there is a $\psi \in \pi_2(f^n(\bar{x}))$ such that $\psi \succ \varphi$. A typical choice for $\succ$ is the multiset extension of a complete simplification ordering on the atoms. If f^n does not apply to $\bar{x}$, $f^n(\bar{x}) = (\emptyset, \emptyset)$.

The inference system I and the set of clauses S defining the problem, determine the set of all clauses derivable from S by I: $S_I^* = \bigcup_{k \geq 0} I^k(S)$, where $I^0(S) = S$, $I(S) = S \cup \{\varphi |\ \varphi \in \pi_1(f(\varphi_1, \ldots, \varphi_n))\ for\ f \in I,\ \varphi_1, \ldots, \varphi_n \in S\}$, and $I^k(S) = I(I^{k-1}(S))$ for all $k \geq 1$. This set is called the *closure* of S with respect to I.

In addition to an inference system I, a *parallel strategy* features a system M of *communication operators*. The basic communication operators are *send* and *receive*. We define them in such a way that communication steps are homogeneous with inference steps, and can be included explicitly in the derivation. Therefore, *send* and *receive* are functions, which take as argument the tuple of clauses being communicated, and return a set of clauses to be added and a set of clauses to be deleted: $send\colon \mathcal{L}_\Theta^* \rightarrow \mathcal{P}(\mathcal{L}_\Theta) \times \mathcal{P}(\mathcal{L}_\Theta)$ and $receive\colon \mathcal{L}_\Theta^* \rightarrow \mathcal{P}(\mathcal{L}_\Theta) \times \mathcal{P}(\mathcal{L}_\Theta)$, such that for all $\bar{x} \in \mathcal{L}_\Theta^*$, $send(\bar{x}) = (\emptyset, \emptyset)$ and $receive(\bar{x}) = (\bar{x}, \emptyset)$. These properties describe the effect of send/receive on the database of the process that executes the operation: *receive* adds the clauses to the database of the receiver, *send* does not modify the database of the sender.

The other major component of a strategy is the *search plan*, which chooses the inference rule and the premises at each step of a derivation. A parallel search plan also controls the *communication* and the *subdivision of the search space* among the deductive processes. The state of a derivation is usually the set (or multiset) of generated and retained clauses. Depending on the strategy, the state can be a tuple of sets. Thus, we use $States$ for the set of possible states and $States^*$ for sequences of states.

For the selection of the next step, the search plan features a *rule-selecting function* and a *premise-selecting function.* In order to account for communication, a parallel search plan may select a communication operator, rather than an inference rule. The selections are made based on the partial history of the derivation, the number of processes, and

[1] In the following $\pi_1(x, y) = x$ and $\pi_2(x, y) = y$ are the projection functions.

the identifier of the process that is executing the selection. Therefore, the domain[2] is $States^* \times \mathbb{N} \times \mathbb{N}$. The rule-selecting function is a function $\zeta\colon States^* \times \mathbb{N} \times \mathbb{N} \rightarrow I \cup M$. The premise-selecting function is a function $\xi\colon States^* \times \mathbb{N} \times \mathbb{N} \times (I \cup M) \rightarrow \mathcal{L}_\Theta^*$, with the inference rule or operator chosen by ζ as additional argument, and such that $\xi((S_0, S_1, \ldots, S_i), n, k, f^m) \in S_i^m$ (i.e., it selects clauses in the current state). If ζ selects an inference rule f and ξ selects a tuple of premises $\bar{x}$, the inference step consists of applying f to $\bar{x}$. If ζ selects *send*, the clauses chosen by ξ will be sent. If ζ selects *receive*, there is no selection of clauses, because clauses will be received from another process.

A *subdivision of the search space* is a subdivision of the inferences in the closure S_I^*. Since S_I^* is infinite and unknown, at each stage S_i of a derivation the search plan subdivides the inferences that can be done in S_i. Thus, the subdivision happens *dynamically* during the derivation. We reason that from the point of view of each process p_k, an inference is either *allowed* (assigned to p_k), or *forbidden* (assigned to others). Therefore, we model the subdivision of the search space by distinguishing between allowed and forbidden steps. For this purpose, a parallel search plan includes a *subdivision function* $\alpha\colon States^* \times \mathbb{N} \times \mathbb{N} \times (I \cup M) \times \mathcal{L}_\Theta^* \rightarrow Bool$, which takes the same arguments as ξ, and, in addition, the tuple of premises chosen by ξ. The subdivision function is partial: where defined, α returns $true$ if the process is allowed to perform the step, and $false$ otherwise. (Communication steps may be always allowed.) A *termination-detecting function* $\omega\colon States \rightarrow Bool$, such that $\omega(S) = true$ if and only if S contains the empty clause, completes the definition of search plan: a *parallel search plan* Σ is a 4-tuple $\Sigma = \langle \zeta, \xi, \alpha, \omega \rangle$ with components defined as above.

A parallel search plan needs to be designed in such a way that when ζ and ξ select a certain f and $\bar{x}$, α is defined on their selection. Furthermore, it is desirable that the subdivision function α is *monotonic*, in the sense of not changing indefinitely the status of a step: for all n, k, f, $\bar{x}$, there exists an $i \geq 0$, such that for all $j \geq i$, $\alpha((S_0, \ldots, S_j), f, \bar{x}, n, k) = \alpha((S_0, \ldots, S_{j+1}), f, \bar{x}, n, k)$ (e.g., it is easy to prove that the subdivision criteria used in Clause-Diffusion are partial monotonic functions).

A sequential search plan has only the components ζ, ξ and ω, with $\zeta\colon States^* \rightarrow I$ and $\xi\colon States^* \times I \rightarrow \mathcal{L}_\Theta^*$ [6]. A parallel search plan $\Sigma' = \langle \zeta', \xi', \alpha, \omega \rangle$ is a *parallelization by subdivision* of a sequential search plan $\Sigma = \langle \zeta, \xi, \omega \rangle$, if Σ' selects inferences in the same way as Σ, so that the only cause of different behaviour is the subdivision of the space. Formally, for all sequence $(S_0, S_1, \ldots, S_i)$, if ζ' chooses an inference rule f, it chooses the same rule that ζ would choose if given $(S_0, S_1, \ldots, S_i)$. Similarly, ξ' chooses the same premises that ξ would choose if given $(S_0, S_1, \ldots, S_i)$ and f. A *parallel strategy* $\mathcal{C}' = \langle I, M, \Sigma' \rangle$ is a *parallelization* of a sequential strategy $\mathcal{C} = \langle I, \Sigma \rangle$, if Σ' is a parallelization of Σ.

The parallel derivations generated for processes $p_0, \ldots, p_{n-1}$ by Σ' will differ from the sequential derivation generated by Σ: the difference is made by the subdivision function, which for every process forbids some choices, forcing the process to choose something else. Furthermore, Σ' will also insert in its derivations communication steps, that Σ does not have. Since communication is made necessary by subdivision, the subdivision is the principle that differentiates the parallel searches from the sequential search. There is no requirement that the parallel search "simulates" the sequential search or is otherwise related to it. Such requirements may apply to parallelizations of algorithms (e.g., the Buchberger algorithm [1]) or to fine-grain parallelizations of theorem-proving strategies (e.g., [8]). In coarse-grain parallelizations such as those we are modelling here, the only

[2]We assume that if there are n processes, they are identified by the numbers $0, \ldots n-1$.

requirement of similarity is the one stated above for $\langle\zeta,\xi\rangle$ and $\langle\zeta',\xi'\rangle$. Thus, the parallel derivations may become very different from their sequential counterpart, as the effects of the subdivision of the space accumulate over time.

Parallel derivations

Given a theorem-proving problem S, the *parallel derivation* generated by a parallel strategy $\mathcal{C} = \langle I, M, \Sigma\rangle$, for n processes $p_0, \ldots p_{n-1}$ is a collection of n sequences

$S = S_0^k \vdash_{\mathcal{C}} S_1^k \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} S_i^k \vdash_{\mathcal{C}} \ldots$, for $k \in [0, n-1]$.

For all k and i, if (1) $\omega(S_i^k) = false$ (proof not found), (2) $\zeta((S_0^k, S_1^k, \ldots S_i^k), n, k) = f$ (ζ selects f), (3) either $f = receive$ and $\bar{x}$ is received, or $f \neq receive$ and $\xi((S_0^k, S_1^k, \ldots S_i^k), n, k, f) = \bar{x}$ (ξ selects $\bar{x}$), and (4) $\alpha((S_0^k, S_1^k, \ldots S_i^k), n, k, f, \bar{x}) = true$ (the step is allowed), then $S_{i+1}^k = S_i^k \cup \pi_1(f(\bar{x})) - \pi_2(f(\bar{x}))$ (add generated/received clauses and delete contracted ones).

In this notion of derivation, all choices are made locally: in this sense, the processes are *loosely coupled*. The local derivations are *asynchronous* in general: any two processes p_j and p_k are not expected to be in stage i simultaneously, and the subscripts of the derivations are independent. The definition of derivation makes no assumption on when events occur. It depends on the strategy whether the processes synchronize for communication (e.g., Team-Work), or communicate asynchronously (e.g., Clause-Diffusion).

A strategy is *complete* if it succeeds ($\omega(S_i^k) = true$ for some k and i) whenever the input set is inconsistent. Completeness reduces to *refutational completeness* of the inference system, and *fairness* of the search plan. A sufficient condition for fairness is *uniform fairness*: a derivation $S_0 \vdash S_1 \vdash \ldots S_i \vdash \ldots$ is *uniformly fair* with respect to an inference system I and a redundancy criterion[3] R if $I(S_\infty - R(S_\infty)) \subseteq \bigcup_{j\geq 0} S_j$, where $S_\infty = \bigcup_{j\geq 0}\bigcap_{i\geq j} S_i$ is the set of persistent clauses [2].

For a parallel derivation, assume $S_\infty^k = \bigcup_{j\geq 0}\bigcap_{i\geq j} S_i^k$ and $S_\infty = \bigcup_{k=0}^{n-1} S_\infty^k$. We need to require that for all $f^m \in I$, for all tuple $\bar{x} \in (S_\infty - R(S_\infty))^m$ of persistent non-redundant premises, such that $f^m(\bar{x}) \neq (\emptyset, \emptyset)$, there are a process p_k and a stage i, such that: $\bar{x} \in (S_\infty^k - R(S_\infty))^m$ (all elements of $\bar{x}$ are together in the memory of p_k at some point: *fairness of communication*), and $\forall j \geq i$, $\alpha((S_0^k, \ldots, S_j^k), n, k, f^m, \bar{x}) = true$ (p_k is allowed to perform the inference on $\bar{x}$: *fairness of the subdivision function*). Since the subdivision function is monotonic, we may require that the step is persistently allowed beyond stage i. Finally, the local derivations need to be fair (*local fairness*).

Theorem: *If a parallel derivation satisfies fairness of communication, fairness of the subdivision function and local fairness, then it is uniformly fair.* (The proof is given in the full paper.)

A parallel derivation *propagates redundancy*, if for all clauses φ, if φ is redundant at stage i for some process p_k, then for all processes there is a stage at which φ is redundant. This is not necessary for completeness: uniform fairness with respect to the expansion rules is sufficient for completeness. On the other hand, maximal contraction, and therefore propagation of redundancy, is important in practice (e.g., see various approaches to *distributed global contraction* in [5]).

[3]For reasons of space we refer to [2, 6] for the definition and explanation of redundancy criterion.

Marked search graphs for parallel search

This model is based on representing both the *search space* and the *search process* in a *marked search graph*: the structure of the graph represents the static space of all possible inferences, while the *marking* represents the dynamics of the search. In a sequential derivation, the dynamic part consists of the selections by the search plan and the deletions by contraction. In a parallel derivation, it also includes the *subdivision* of the space and the *communication*.

The static structure of the search space depends on the problem and the inference system, so that it is the same regardless of whether the space is searched in parallel or sequentially. Given an input set S and an inference system I, the *search space* induced by S and I is represented by the *search (hyper)graph* $G(S^*_I) = (V, E, l, h)$, where the vertices in V represent the clauses in the closure S^*_I, and the hyperarcs in E represent the inferences. The l is a *vertex-labelling function* $l\colon V \rightarrow \mathcal{L}_\Theta / \stackrel{\bullet}{=}$, which associates to each vertex the equivalence class of all variants of a clause ($\stackrel{\bullet}{=}$ is equivalence up to variable renaming). The h is an *arc-labelling function* $h\colon E \rightarrow I$ from hyperarcs to inference rules. If there is an inference $f(\varphi_1, \ldots, \varphi_n) = (\{\psi_1, \ldots, \psi_m\}, \{\alpha_1, \ldots, \alpha_p\})$ for $f \in I$, then E contains a hyperarc $e = (v_1, \ldots, v_k; w_1, \ldots, w_p; u_1, \ldots, u_m)$ where (1) $h(e) = f$, (2) $v_1, \ldots, v_k$ are the vertices labelled by the premises that are not deleted (i.e., $l(v_j) = \varphi_j$ and $\varphi_j \notin \{\alpha_1, \ldots, \alpha_p\}$, for all j, $1 \leq j \leq k$, where $k = n - p$), (3) $w_1, \ldots, w_p$ are the vertices labelled by the deleted clauses (i.e., $l(w_j) = \alpha_j$, for all j, $1 \leq j \leq p$), and (4) $u_1, \ldots, u_m$ are the vertices labelled by the generated clauses (i.e., $l(u_j) = \psi_j$, for all j, $1 \leq j \leq m$).

In most cases, we only need hyperarcs where at most one clause is added or deleted. For instance, a *resolution* arc has the form $(v_1, \ldots, v_n; u)$, where u is the resolvent of $v_1, \ldots, v_n$; a *simplification* arc has the form $(v; w; u)$, where v reduces w to u; and a *normalization* arc has the form $(v_1, \ldots, v_n; w; u)$, where u is a normal form of w with respect to the simplifiers $v_1, \ldots, v_n$. Contraction inferences that purely delete clauses are represented as replacement by *true*, where *true* is a dummy clause, such that $true \prec \varphi$ for all φ. A special vertex T in the search graph is labelled by *true*.

Given the search graph $G(S^*_I) = (V, E, l, h)$, the representation of the search process during a derivation needs to cover: (1) the selections by the search plan, (2) the deletions by contraction, (3) the subdivision of the search space, (4) the effect of communication. Note that these four aspects are all intertwined, because it is the search plan that decides contractions, subdivision and communications, and these in turn affect the successive choices of the search plan. The search process is captured by *marking functions* for vertices and arcs. For each process p_k there is a *vertex-marking function* $s^k\colon V \rightarrow Z$ such that: $s^k(v) = m$, if m variants of $l(v)$ are present for process p_k, $s^k(v) = -1$, if all variants of $l(v)$ have been deleted by p_k, and $s^k(v) = 0$ otherwise. The vertex-marking function represents the dynamic deletions by contraction (2) and the consequences of the communication steps (4): if a clause is deleted, its marking becomes negative; if a clause is received, its marking is incremented. For the arcs, for each process p_k there is an *arc-marking function* $c^k\colon E \rightarrow \mathbb{N} \times Bool$ defined as follows[4]: $\pi_1(c^k(e)) = m$, if p_k executed arc e m times, $\pi_2(c^k(e)) = true/false$, if p_k is allowed/forbidden to execute e. The first component of the arc-marking function represents the selections done by the search plan (1); the second component, called *permission marking*, represents the subdivision of the space (3).

By using multiple marking functions we can represent the effects of all the processes on the same search graph: a *parallel marked search-graph* $(V, E, l, h, \bar{s}, \bar{c})$ is given by a

[4]*Bool* also contains $\perp$, so that the permission marking of an arc can be undefined.

search graph, an n-tuple $\bar{s}$ of vertex-marking functions, and an n-tuple $\bar{c}$ of arc-marking functions.

The next goal is to represent the evolution of the search space during a derivation. For this purpose, we define the *pre-conditions* and *post-conditions* of a step: a hyperarc $e = (v_1, \ldots, v_n; v_{n+1}; v_{n+2})$ is *enabled* at process p_k, if (1) $s^k(v_j) > 0$ for all $j \leq n+1$ (all premises are present), and (2) $\pi_2(c^k(e)) = \mathit{true}$ (the step is allowed). An operation $\mathit{send}(\bar{x})$, where $\bar{x} = (\varphi_1, \ldots \varphi_n)$, is *enabled* at p_k if $s^k(v_j) > 0$ for all j, $l(v_j) = \varphi_j$, $0 \leq j \leq n$. An operation $\mathit{receive}(\bar{x})$ is enabled without conditions.

For the post-conditions, if p_k executes an enabled hyperarc $e = (v_1, \ldots, v_n; v_{n+1}; v_{n+2})$ the effect is to decrement the marking of the deleted clause (vertex v_{n+1}: if the marking is 1, it goes to -1, denoting that the last variant has been deleted), and to increment the marking of the generated clause (vertex v_{n+2}: if the marking is -1, it becomes 1, denoting that a deleted clause has been regenerated). There is no post-condition for sending clauses, while the post-condition of receiving clauses is to increment their markings like for generation.

Then, we associate to a parallel derivation n *successions of vertex-marking functions* $\{s^k_i\}_{i\geq 0}$, one for each process p_k, and n *successions of arc-marking functions* $\{c^k_i\}_{i\geq 0}$, one for each process p_k. In the initial state, $s^k_0(v)$ is 1 for all p_k if $\varphi = l(v)$ is an input clause, 0 otherwise. Alternatively, input clauses may be given marking 1 only at one process, say p_0, which is responsible for pre-processing and broadcasting them. For all successive stages $i \geq 0$, $s^k_{i+1}(v)$ is modified as described above, if p_k executes an enabled hyperarc which includes v, or the clause of v is received, and it is left unchanged otherwise. For the arc-markings, at the initial stage, for all $a \in E$, $\pi_1(c^k_0(a)) = 0$. At all subsequent stages $i \geq 0$, $\pi_1(c^k_{i+1}(a)) = \pi_1(c^k_i(a)) + 1$ if a is enabled at stage i and p_k executes it, and it is unchanged otherwise. For the second component, $\pi_2(c^k_i(a)) = \alpha((S_0, \ldots S_i), n, k, f, \bar{x})$, where f and $\bar{x}$ are the inference rule and the premises of arc a. Since α is monotonic, the permission marking of the arcs is also monotonic, that is, it stabilizes eventually.

Note that inference steps performed by p_k affect only the markings s^k and c^k: this property mirrors the fact that the databases of the deductive processes are separate. The *generated search space* up to stage i is determined by the vertices with non-zero marking.

Discussion

A key feature of this approach to the modelling of search is the ability to represent search spaces made dynamic by the contraction inferences, the subdivision of the space, and the communication steps. All three these aspects cannot be represented by structural modifications of the graph (e.g., deletions for contraction and subdivision, and additions for communication). Such an approach would make the structure of the search space dependent on the search plan, and it would make impossible to represent the parallel search processes on the same graph.

This work will continue with the extension of the measurement of search complexity in [6] to capture both the advantages and the overhead of parallelism, and with results in the comparison of strategies.

References

[1] G. Attardi and C. Traverso. A strategy-accurate parallel Buchberger algorithm. In Hoon Hong, editor, *Proc. of the 1st PASCO Symposium*, volume 5 of *Lecture Notes*

Series in Computing, pages 22–33. World Scientific, 1994.

[2] L. Bachmair and H. Ganzinger. Non-clausal resolution and superposition with selection and redundancy criteria. In *Proc. of LPAR-92*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 273–284. Springer Verlag, 1992.

[3] M. P. Bonacina. On the reconstruction of proofs in distributed theorem proving: a modified Clause-Diffusion method. *J. of Symbolic Computation*, 21:507–522, 1996.

[4] M. P. Bonacina and J. Hsiang. Parallelization of deduction strategies: an analytical study. *J. of Automated Reasoning*, 13:1–33, 1994.

[5] M. P. Bonacina and J. Hsiang. The Clause-Diffusion methodology for distributed deduction. *Fundamenta Informaticae*, 24:177–207, 1995.

[6] M. P. Bonacina and J. Hsiang. On the representation of dynamic search spaces in theorem proving. In C.-S. Yang, editor, *Proc. of the Int. Computer Symposium*, pages 85–94, 1996. Full version: "On the modelling of search in theorem proving – Towards a theory of strategy analysis", Tech. Rep., Dept. of Comp. Sci., Univ. of Iowa, 1995.

[7] R. Bündgen, M. Göbel, and W. Küchlin. A master-slave approach to parallel term-rewriting on a hierarchical multiprocessor. In J. Calmet and C. Limongelli, editors, *Proc. of the 4th DISCO Symposium*, volume 1128 of *Lecture Notes in Computer Science*, pages 184–194. Springer Verlag, 1996.

[8] R. Bündgen, M. Göbel, and W. Küchlin. Strategy-compliant multi-threaded term completion. *J. of Symbolic Computation*, 21(4–6):475–506, 1996.

[9] J. Denzinger and S. Schulz. Recording and analyzing knowledge-based distributed deduction processes. *J. of Symbolic Computation*, 21(4–6):523–541, 1996.

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