Towards a Parallel Search Engine for Planning Systems Based on Linear Time Logic

Marta Cialdea, Carla Limongelli, Andrea Orlandini, Valentina Poggioni

Dipartimento di Informatica e Automazione
Università degli Studi di Roma Tre
Via della Vasca Navale, 79 I-00146 – Roma
e-mail: {cialdea,limongel,orlandin,poggioni}@dia.uniroma3.it

Abstract. A planning problem can be entirely encoded as a set of linear temporal logic (LTL) formulae, in such a way that planning is reduced to model search. In order for this approach to be effective, it is important to enhance the performances of LTL provers. In this work, we study a parallel algorithm for LTL model search, based on the tableaux calculus, and present the first promising experiments with its implementation on a cluster of non homogeneous machines.

1 Introduction

Modal temporal logics are useful tools for modeling and reasoning about time, and their application to planning has been proposed in different formats: planning via deduction in interval-based temporal logics [18, 14], planning via model checking [11, 12, 4], and planning via model construction in linear temporal logic [3, 10]. Moreover, temporal logic has been used to encode control knowledge and improve on a specialized planning algorithm [1, 2].

Based on the theoretical framework of [6], [10] presents a planner fully based on LTL, in the "planning as satisfiability" approach: a planning problem is entirely encoded as a set of LTL formulae and planning is reduced to model search. In [8, 9], the approach is extended to contingency planning.

The main advantages of using LTL come from its great expressive power. In fact, there is no need to represent time points explicitly, neither to fix a bound to the plan length in advance. The expressive power of LTL allows one also to represent domain restrictions in the style of [7], intermediate tasks, like in [1], domain and control knowledge, as well as temporally extended goals.

Moreover, the logical encoding of planning problems provides a formal semantics of the planning languages and makes it possible to implement useful tools aiming at pointing out possible inconsistencies or redundancies in the specification.

The kernel of the system presented in [10] is constituted by a model search engine based on tableaux [19], which can be given an efficient implementation thanks also to the use of Ordered Binary Decision Diagrams (OBDD). Due to

the heavy computational complexity of LTL, however, its application to planning would greatly benefit from an intensive exploitation of the available computational resources. This work explores a parallel approach to tableaux based theorem proving for LTL and presents the first promising experiments with a prototype system.

Parallel and distributed approaches to automated theorem proving have been widely investigated (see [5, 17]). In particular in [13] an analysis of a tableaux method for LTL identifies different forms of potential parallelism that can be managed by concurrently cooperating *heterogeneous* agents. The approach presented here is different: it is based on the "divide et impera" approach: a task in tableaux construction is identified that can be split into smaller *homogeneous* processes. The parallelization acts during the construction of each time state: the set of formulas to be expanded is split into k disjoint subsets (where k is the number of processes), the k tableaux expansions are carried out in parallel, and the k results are suitably combined.

This parallelization technique has been already described in [15], together with results coming from a simulation of the parallelization. Such results, however, were not completely significant, since the communication overhead could not be taken into account. In this paper we show the results obtained from "worst case" experiments, i.e. carried on a cluster of non homogeneous machines.

2 Preliminaries

In this section we briefly introduce the syntax and semantics of LTL, recall how a planning problem is encoded so that planning can be reduced to model search, and describe the tableaux calculus for LTL and a sequential algorithm for model search.

We consider the language of propositional LTL, containing the future time operators $\Box, \Diamond$ and $\bigcirc$: $\Box A$ means that A is true now and will always be true, $\Diamond A$ that A is either true now or sometime in the future, and $\bigcirc A$ that A holds in the next state. The model of time underlying LTL is a denumerably infinite sequence of states, a temporal frame, that can be identified with $\mathbb{N}$. Its elements are called time points, and an interpretation $\mathcal{M}$ is a function mapping each time point i to the set of propositional letters true at i.

The satisfiability relation $\mathcal{M}_i \models A$, for $i \in \mathbb{N}$, is defined as follows:

1. $\mathcal{M}_i \models p$ iff $p \in \mathcal{M}(i)$, for any propositional letter p in the language.
2. $\mathcal{M}_i \models \neg A$ iff $\mathcal{M}_i \not\models A$.
3. $\mathcal{M}_i \models A \wedge B$ iff $\mathcal{M}_i \models A$ and $\mathcal{M}_i \models B$.
4. $\mathcal{M}_i \models A \vee B$ iff either $\mathcal{M}_i \models A$ or $\mathcal{M}_i \models B$.
5. $\mathcal{M}_i \models \Box A$ iff for all $j \geq i$, $\mathcal{M}_j \models A$.
6. $\mathcal{M}_i \models \Diamond A$ iff there exists $j \geq i$ such that $\mathcal{M}_j \models A$.
7. $\mathcal{M}_i \models \bigcirc A$ iff $\mathcal{M}_{i+1} \models A$.

Truth is satisfiability in the initial state: a formula A is true in $\mathcal{M}$ (and $\mathcal{M}$ is a model of A) iff $\mathcal{M}_0 \models A$. Truth of sets of formulae is defined as usual.

A temporal interpretation describes how the world evolves in time, hence a suitable set of formulae can constrain the behaviour of the world and be taken as the specification of a planning domain. The notion of a temporal interpretation satisfying all the constraints of a planning problem, thus representing a plan that solves the problem, is quite straightforward: basically, the initial state satisfies the (classical) encoding of the initial state S_0 and there exists a state g satisfying the (classical) formula *Goal* representing the goal. Actions are represented by atomic formulae: if an action is executed at the i-th step, its representation a holds at time point i, and its preconditions hold at i as well; moreover, classical effect and frame conditions for actions are satisfied at any time point. If we assume that the initial state description is complete, it can be shown that $P = \langle A_0, ..., A_n \rangle$ is a plan solving the planning problem Π iff there exists an interpretation $\mathcal{M}$ for a propositional language $\mathcal{L}$, satisfying all the constraints of Π, that determines P [10]. Hence, plans actually solving a problem can be characterized in terms of temporal models.

The LTL specification of the initial state, goal, action preconditions and incompatibility relations described in [10] produces the initial LTL formula F_0 that describes the problem and the planning domain.

Example Let us consider a planning problem where an agent has to pass a door that is initially closed. The LTL encoding of such a problem is the conjunction of the following formulae:

$$\Box(do\_pass \rightarrow open),$$
$$\Box(do\_open \rightarrow \neg open),$$
$$\Box(\bigcirc in \equiv (in \wedge \neg do\_pass) \vee (\neg in \wedge do\_pass)),$$
$$\Box(\bigcirc open \equiv open \vee do\_open),$$
$$\neg in \wedge \neg open,$$
$$\Diamond in$$

A possible model of the encoding is the sequence of time points where *do_open* is true at state 0, *do_pass* is true at state 1 and *in* is true at state 2. The corresponding plan is made up of the sequence of actions $\langle do\_open, do\_pass \rangle$.

Model search can be performed by means of the tableaux calculus presented in [19], that is based on the following set of expansion rules acting on sets of formulas (the nodes of the tableaux):

$$\frac{A \wedge B, S}{A, B, S}\ (\alpha) \qquad \frac{A \vee B, S}{A, S \mid B, S}\ (\beta)$$

$$\frac{\Box A, S}{A, \bigcirc\Box A, S}\ (\nu) \qquad \frac{\Diamond A, S}{A, S \mid \bigcirc\Diamond A, S}\ (\pi)$$

$$\frac{\bigcirc A_1, ..., \bigcirc A_n, \ell_1, ...\ell_m}{A_1, ..., A_n}\ (\bigcirc)$$
$$where\ \ell_1, ...\ell_m\ are\ literals$$

The first four rules are called "static" and generate "static expansions" (their premises and conclusions refer to the same time point), while the last one is

"dynamic": it generates the description of a new time point, the successor of the time point described in the premise.

When the next-state rule ($\bigcirc$) is applicable to a node N labelled by S, generating S', loop-checking is performed: if a node N' with the same label S' already exists in the tableau, then S' is not created but an edge is added from N to N'. Hence a tableau, in this approach, is not a tree but a graph. As usual, if a node contains a pair of complementary literals, it is closed.

A looping branch represents a cyclic interpretation. However, not every looping branch is open. In order to ensure that it represents a model of the initial set of formulae, it must be checked that all "eventualities" are satisfied, i.e. that for every formula of the form $\Diamond A$ occurring in some node N of the branch, there is a path in the branch leading from N to a node N' containing the formula A.

The sequential algorithm for model search that we describe next is based on a mixed depth/breadth first search: given a node to be expanded, first of all its static expansions are built, till a set of nodes which can be expanded by means of the next-state rule is obtained. Then the next-state rule is applied to one of such nodes to create a new time point, that will be then expanded further on. When a closed branch is found the algorithm backtracks and looks for a model on another branch. A sketch of the sequential algorithm is given in figure 1.

EXPAND (F^0) **:** **LOOP**(F^0)

LOOP (F^t) **:**

1. (EXPAND:) the static expansion rules are applied to F^t, until the next-state rule can be applied. Let $\{R_i^t \mid \quad i = 1, \ldots, n\}$ be the set of all the *open* leaves of the static expansion of the node F^t, obtained at this stage (the inconsistent R_i^t are eliminated after the expansion).
2. (CHOOSE NEXT NODE:) one of the R_i^t is chosen (*choice point*):
   if R_i^t is a set of literals
   . **then SUCCESS** (a model is found)
   . **else** R_i^t is expanded by application of the next-state rule, generating F_i^{t+1}
3. (CHECKS:)
   if F_i^{t+1} contains complementary literals
   . **then** FAIL
   . **else if** F_i^{t+1} is already present in a node of the branch from F^0 to F_i^t
   . **then**
   . **if** all eventualities are satisfied
   . **then** SUCCESS (a model is found)
   . **else** FAIL
   . **else** **LOOP**(F^{t+1});

Fig. 1. The sequential backtracking algorithm for tableaux expansion. When the algorithm fails, it backtracks to the (CHOOSE NEXT NODE) step. When the algorithm succeeds, it returns the last node found, that has a pointer to the father. In this way the path to the root can be reconstructed. If **LOOP**(F^0) fails, no model exists for F^0.

This proof procedure terminates: in fact a branch of the tableau terminates either because there are no further formulas to expand, or because it contains a pair of complementary literals, or because a loop has been detected. In the latter case, eventuality satisfaction is checked.

3 Parallelization Technique

In this section we show how the sequential algorithm can be parallelized.

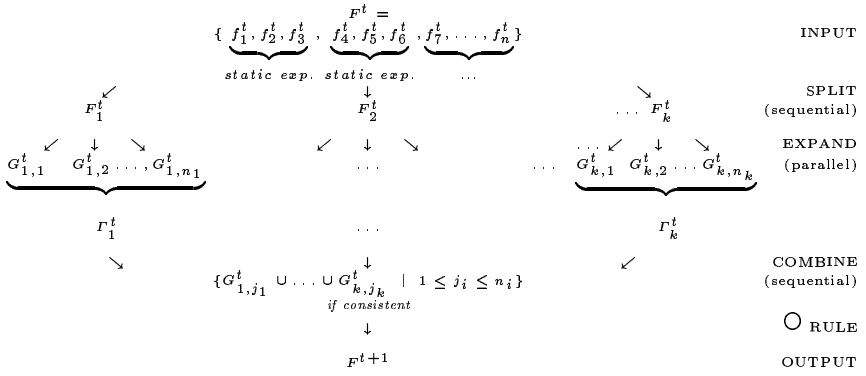
The approach is based on the idea of distributing the formulas to be expanded over k processors and expanding in parallel each of the obtained subsets. A suitable reconstruction step based on a lazy approach gives then the final result. More precisely, the set of formulas F^t is split into k subsets $F_1^t, \ldots, F_k^t$. The k sets of static expansions are performed in parallel (by applying only static expansion rules), each producing a set of nodes (sets of formulas) G_i^t (with associated eventualities) where

$$G_i^t = \{g_{i,1}^t, \ldots, g_{i,n_i}^t\}, \quad i = 1, \ldots, k$$

These results are combined producing:

$$C = combine(G_1^t, \ldots, G_k^t) = \{g_{1,j_1}^t \cup \ldots \cup g_{k,j_k}^t \mid 1 \leq j_i \leq n_i\}$$

From this set of nodes an element $R^t \in C$ is chosen (*choice point*). If R^t does not contain complementary literals, it is expanded by means of the next-state rule, generating the next state F^{t+1}. The process is repeated for F^{t+1} until either a model is found or the search fails. A pictorial representation of the expansion procedure is given below.



In figure 2 we show the pseudocode of the parallel algorithm pointing out the difference between the parallel and the sequential part.

Note that the part of the algorithm actually parallelized is the static expansion of nodes.

P_EXPAND (F^0, k) **: P_LOOP**(F^0, k)
P_LOOP (F^t, k) **:**

1. (SPLIT:) F^t is partitioned into $F_1^t, \ldots, F_k^t$, in such a way that for all $i, j = 1, \ldots, k$, the sizes of F_i and F_j differ at most by 1.
2. (EXPAND:) the static expansions of each $F_i^t,\ i = 1, \ldots, k$ are built in parallel. Let $\Gamma_1^t, \ldots, \Gamma_k^t$ be the results of each process (each Γ_i^t is a set of *open* nodes).
3. (COMBINE:) The set $\Sigma = combine(\Gamma_1^t, \ldots, \Gamma_k^t)$ is (lazily) constructed.
4. (CHOOSE NEXT NODE:) one element R^t of Σ is chosen (*choice point*):
   if R^t is a set of literals
   . **then if** it is consistent
   . **then** SUCCESS (a model is found)
   . **else** FAIL
   . **else** R^t is expanded by application of the next-state rule, generating F^{t+1}
5. (CHECKS:)
   if F^{t+1} contains complementary literals
   . **then** FAIL
   . **else if** F^{t+1} is already present in node of the branch from F^0 to F^t
   . **then if** all eventualities are satisfied
   . **then** SUCCESS (a model is found)
   . **else** FAIL
   . **else P_LOOP**(F^{t+1}, k);

Fig. 2. The Parallel Algorithm.

4 Experimental Results

The algorithm has been implemented in C, where parallelization is managed in the PVM framework, and tested over a non homogeneous cluster composed by (up to) 5 mono-processors PC (Apple Powerbooks and PC compatibles) connected in a Ethernet net (10 Mb/s). Here follows a table with information about the machines.

machine	processors	ram	O.S.
Apple Powerbook	Motorola PowerPC G4@1GHz	256Mb	MacOSX 10.3
Apple Powerbook	Motorola PowerPC G4@1GHz	256Mb	MacOSX 10.3
Apple Powermac	Motorola PowerPC G4@400MHz	320Mb	MacOSX 10.3
PC compatibile	Intel Penthium4@1.5GHz	256Mb	LinuxMandrake 9.1
PC compatibile	Intel Penthium3@450MHz	256Mb	LinuxMandrake 9.1

This kind of cluster does not match the best configuration since slower machines affect the behaviour of the better ones. Generally best performances are obtained with homogeneous clusters.

The experimental results we are going to show are based on the most significant measures for parallel experiments: speed up and efficiency. *Speed up* measures the actual advantage of the parallel execution with a given number p

of processors ($T_{par,p}$), w.r.t. the sequential one (T_{seq}):

$$S_p = \frac{T_{seq}}{T_{par,p}}$$

Obviously the ideal speed up is exactly p.

The measure of how much a single processor is involved into a computation is the *efficiency*:

$$E = \frac{S_p}{p}$$

Let us note that $E = 1$ means the best use of all the processors.

According to Amdahl's law, the speed up is not linear when the number of processors increase, but it tends to saturate and the efficiency drops. The ability to maintain efficiency at a fixed value by simultaneously increasing the number of processors and the size of the problem is called *scalability* and parallel systems with this ability are called *scalable*. The scalability of a parallel system reflects its ability to use increasing resources effectively.

The algorithm has been tested for problems with 20 formulas, 10 atoms and depth 10 and with 30 formulas 20 atoms and depth 10. The test formulas have been randomly generated and they respect the requirements given by TANCS [16] (in particular the formulas for the modal logic S4). The original code for generating the formulas has however been modified, since in its original formulation satisfiable formulae have always very short models.

For each configuration we have computed the average time given by 35 different random problems (with 20 and 30 formulas). For each problem we have considered the time given by the average computing time of four executions. In fact different executions on the same problem can take different times because of unpredictable network traffic.

Whichever the number of computing machines (1 to 5), the parallel phases of the algorithm are always split over five processes, which are distributed by PVM among the available processors in a transparent way. In fact, our main interests was to see whether taking the communications time into considerations would invalidate the positive results obtained in [15].

The average values of the execution times with different numbers of processors are shown in the table that follows. All the times reported below are measured with a precision of 0.001 secs. The first row of the table refers to the sequential algorithm, run on one of the faster machines. The second row calls 5 processes on a single machine, one of the faster ones. The machines in the cluster are then added in order of decreasing computational power.

n.of.processors	20 formulas	30 formulas
1 sequential	20,539 sec	52,855 sec
1	6,024 sec	27,552 sec
2	5,390 sec	23,032 sec
3	4,003 sec	21,898 sec
4	3,735 sec	20,748 sec
5	4,815 sec	21,916 sec

First of all let us note the dramatical improvement of the parallel algorithm running over one processor, but split over five processes, w.r.t. the sequential algorithm. The *divide et impera* technique alone produces a great improvement of the execution time.

The improvement with 2, 3, 4 and 5 processors is considerably slower (in the last case the execution times are even longer, because of the addition of the slowest machine). A reason for this, beyond the nature of the cluster, is the fact that, since memory is not shared, there is a big amount of communication data, i.e. formulas, that have to be manipulated and exchanged among all the processors.

The results show however that the heavy communication overhead does not exceed benefits coming from parallelization. In fact, if we consider the speed up of the algorithm running over one processor w.r.t. the runs with more processors, we obtain a positive trend of the speed up, except for the last measure that is due to the addition of the slowest PC to the cluster:

		$k=2$	$k=3$	$k=4$	$k=5$
20 from.	$\frac{T_{par,1}}{T_{par,k}}$	1.118	1.505	1.613	1.251
30 form.	$\frac{T_{par,1}}{T_{par,k}}$	1.196	1.259	1.328	1.258

The speed up measures given in figure 3 – and related now to T_{seq} – are based on the geometric mean, generally taken as a reasonable measure of the system's behaviour:[1]

$$\overline{s}_g = \sqrt[n]{s_1 \dots s_n} \text{ (geometric mean)}$$

Here $s_1, \dots, s_n$ are the speeds up of n random problems.

Figure 3 shows the results for problems with 30 formulas.

The speed up values are quite high, this means that the parallelization is always profitable because every measure is tending to the ideal speed up.

Let us consider now the system scalability and efficiency, considering problems with 20 formulas run with 1, 2 and 3 processors and problems with 30 formulas run with 4 and 5 processors.

Figure 4 shows that the efficiency oscillates around a value close to 1 and that the speed up trend is almost linear.

5 Conclusions

Even if the cluster used for testing the algorithm is not homogeneous, the experimental results show a clear advantage of the parallelization and a good behaviour of the algorithm especially from the point of view of scalability.

[1] The results obtained considering, in alternative, the waiting times mean introduced in [17], often used for testing theorem provers, are not significantly different.

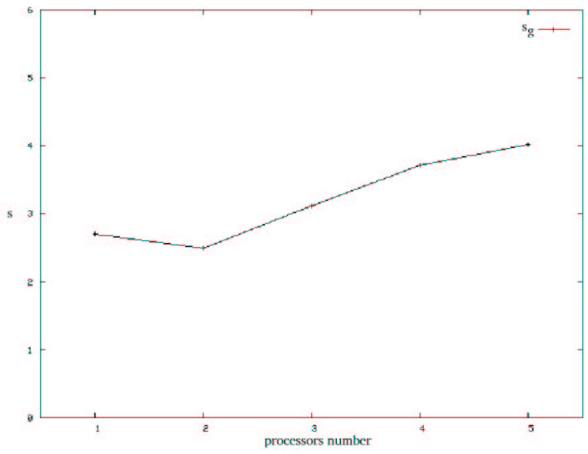


Fig. 3. Speed up related to geometric means for problems with 30 formulas.

In particular the comparison between the two versions of the algorithm (sequential and parallel over a single processor), shows a better behaviour of the parallelized version. Howewer, in order to confirm and give a clearer evaluation of the possible benefits of the *divide et impera* approach alone, the algorithm must be re-implemented using standard optimization techniques. In fact, such techniques may influence the sequential and parallel versions in different degrees, so that, from this point of view, a prototypal version cannot give the information we need.

The optimization of the implementation, moreover, will make it possible to collect experimental results over significant planning domains.

In the next future we are planning to collect new experimental results from a bigger and homogeneous cluster, as well as on a parallel architecture with shared memory. In the latter case, the problem arising from the big amount of formulas that have to be packed (and unpacked) when exchanged among the processors should be overcome.

We are confident that the proposed parallelization technique over an appropriate parallel architecture could significantly improve the efficiency of the "planning as satisfiabilty in LTL" approach and will enlarge the fields of application.

References

1. F. Bacchus and F. Kabanza. Planning for temporally extended goals. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1215–1222. AAAI Press / The MIT Press, 1996.
2. F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 16:123–191, 2000.

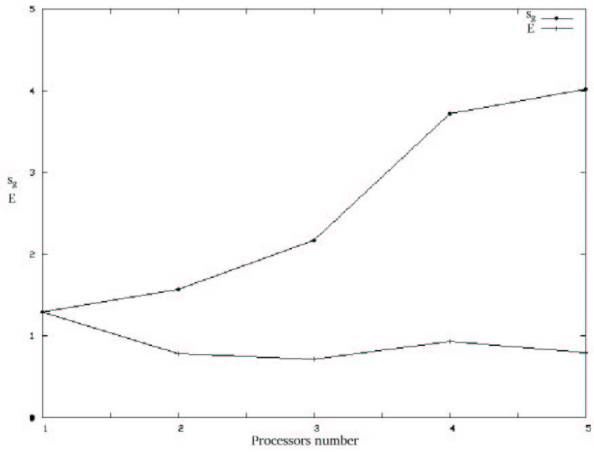


Fig. 4. Scalability and efficiency.

3. H. Barringer, M. Fisher, D. Gabbay, and A. Hunter. Meta-reasoning in executable temporal logic. In *Proc. of the Second Int. Conf. on Principles of Knowledge Representation and Reasoning*, 1991.
4. P. Bertoli, A. Cimatti, Pistore M., and P. Traverso. A framework for planning with extended goals under partial observability. In *13-th International Conference onAutomated Planning and Scheduling (ICAPS'03)*. AAAI Press, 2003.
5. M.P. Bonacina. A taxonomy of parallel strategies for deduction. *Annals of Mathematics and Artificial Intelligence*, 29(1-4):223–257, 2000.
6. S. Cerrito and M. Cialdea Mayer. Using linear temporal logic to model and solve planning problems. In F. Giunghiglia, editor, *Proceedings of the 8th International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'98)*, pages 141–152. Springer, 1998.
7. A. Cesta and A. Oddi. DDL.1: a formal description of a constraint representaton language for physical domains. In M. Ghallab and A. Milani, editors, *New Direction in AI Planning*, pages 341–352. IOS Press, 1996.
8. M. Cialdea Mayer and C. Limongelli. Linear time logic, conditioned models and planning with incomplete knowledge. In C. Fermüller and U. Egly, editors, *Proc. of the Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 2381 of *LNAI*, pages 70–84. Springer, 2002.
9. M. Cialdea Mayer, C. Limongelli, A. Orlandini, and V. Poggioni. Planning under uncertainty in linear time logic. In F. Turini A. Cappelli, editor, *Advances in Artificial Inteligence AI*IA 2003*, number 2829 in LNAI, pages 324–335. Springer, 2003.
10. M. Cialdea Mayer, A. Orlandini, G. Balestreri, and C. Limongelli. A planner fully based on linear time logic. In S. Chien, S. Kambhampati, and C.A. Knoblock, editors, *Proc. of the 5th Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS-2000)*, pages 347–354. AAAI Press, 2000.
11. A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: a decision procedure for $\mathcal{AR}$. In S. Steel and R. Alami, editors, *Proc. of*

the Fourth European Conference on Planning (ECP-97), pages 130–142. Springer-Verlag, 1997.
12. A. Cimatti and M. Roveri. Conformant planning via model checking. In *Proc. of the Fifth European Conference on Planning (ECP-99)*, 1999.
13. R. Johnson. Communicating agents for concurrent temporal tableaux. In *Theorem Proving with Analytic Tableaux and Related Methods. 4th International Workshop, TABLEAUX-95*, volume 918 of *LNCS*. Springer Verlag, 1995.
14. J. Koehler and R. Treinen. Constraint deduction in an interval-based temporal logic. In M. Fisher and R. Owens, editors, *Executable Modal and Temporal Logics, (Proc. of the IJCAI'93 Workshop)*, volume 897 of *LNAI*, pages 103–117. Springer, 1995.
15. C. Limongelli, A. Orlandini, and V. Poggioni. A parallel computation technique for linear time logic tableaux. In F. Pirri M. Cialdea Mayer, editor, *Tableaux 2003, Position Papers and Tutorials*. Aracne, 2003.
16. F. Massacci and F.M. Donini. Design and results of TANCS-00. In R. Dyckhoff, editor, *Proceedings of the Fourth International Conference on Analytic Tableaux and Related Methods (TABLEAUX 2000)*, volume 1847 of *LNAI*, pages 52–56. Springer-Verlag, 2000.
17. J. Schumann. Sicotheo: Simple competitive parallel theorem provers. In *Conference on Automated Deduction*, pages 240–244, 1996.
18. B. Stephan and S. Biundo. Deduction based refinement planning. In B. Drabble, editor, *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pages 213–220. AAAI Press, 1996.
19. P. Wolper. The tableau method for temporal logic: an overview. *Logique et Analyse*, 28:119–152, 1985.