A Tableau System for Right Propositional Neighborhood Logic over Finite Linear Orders: an Implementation

D. Bresolin[1], D. Della Monica[2], A. Montanari[3], and G. Sciavicco[4]

[1] Department of Computer Science, University of Verona
Verona, Italy (davide.bresolin@univr.it)
[2] ICE-TCS, School of Computer Science, University of Reykjavik
Reykjavik, Iceland (dariodm@ru.is)
[3] Department of Mathematics and Computer Science, University of Udine
Udine, Italy (angelo.montanari@dimi.uniud.it)
[4] Department of Information, Engineering and Communications
University of Murcia, Murcia, Spain(guido@um.es)

Abstract. Interval temporal logics are quite expressive temporal logics, which turn out to be difficult to deal with in many respects. Even finite satisfiability of simple interval temporal logics presents non-trivial technical issues when it comes to the implementation of efficient tableau-based decision procedures. We focus our attention on the logic of Allen's relation *meets*, a.k.a. Right Propositional Neighborhood Logic (RPNL), interpreted over finite linear orders. Starting from a high-level description of a tableau system, we developed a first working implementation of a decision procedure for RPNL, and we made it accessible from the web. We report and analyze the outcomes of some initial tests.

1 Introduction

Propositional interval temporal logics play a significant role in computer science, as they provide a natural framework for representing and reasoning about temporal properties in a number of application domains. This is the case, for instance, with natural language understanding and processing, where significant interval-based logical formalisms have been developed to represent and to reason about tenses and temporal prepositions, e.g., [14,16]. As another example, the possibility of encoding and reasoning about various constructs of imperative programming languages in interval temporal logic has been systematically explored by Moszkowski [15]. Other meaningful applications of interval temporal logics can be found in temporal knowledge representation and reasoning, systems for temporal planning and maintenance, qualitative reasoning, theories of action and change, specification, verification, and design of hardware components, concurrent real-time processes, event modeling, temporal aggregation in databases, and computational linguistics [9,12].

Interval logic modalities correspond to relations between (pairs of) intervals. In particular, Halpern and Shoham's modal logic of time intervals HS [13] features a set of modalities that make it possible to express all Allen's interval relations [1]. HS turns out to be undecidable over all meaningful classes of linear orders, including the class of finite linear orders we are interested in. Temporal reasoning on finite linear orders comes into play in a variety of areas. This is the case, for instance, with planning problems, which consist of finding a finite sequence of actions that, applied to an initial state of the world, leads to a goal state within a bounded amount of time, satisfying suitable conditions about which sequence of states the world must go through. In the last years, a lot of work has been done on (un)decidability and complexity of HS fragments. The complete picture about finite linear orders is given in [6]: there exist 62 non-equivalent decidable fragments of HS, partitioned into four complexities classes, ranging from NP-complete to non-primitive recursive. For each decidable fragment, an optimal decision procedure has been devised. Nevertheless, none of them is available as a working system (they are declarative procedures, which turn out to be unfeasible in practice), with the only exception of the logic of subintervals D over dense linear orders [3]. D has been implemented in LoTrec [10], a generic theorem prover that allows one to specify the rules for his/her own modal/temporal logic. Unfortunately, LoTrec is in general not suitable for interval logics (D over dense linear orders is a very special case), because (i) it does not support the management of world labels explicitly, and (ii) it does not admit closing conditions based on the number of worlds generated in the construction of a tentative model, but only closing conditions based on patterns and repetitions.

In this paper, we focus our attention on one of the simplest decidable fragments of HS, namely, Right Propositional Neighborhood Logic (RPNL) [4,8,11], interpreted on finite linear orders, whose satisfiability problem has been proved to be NEXPTIME-complete. RPNL features a single modality corresponding to Allen's relation *meets*. We devised and implemented a working tableau-based decision procedure for RPNL, based on the original (non-terminating) tableau system given in [11], which exploits the small model theorem proved in [8] to guarantee termination.

2 Syntax and semantics of RPNL

Let $\mathbb{D} = \langle D, < \rangle$ be a finite linear order. An *interval* over $\mathbb{D}$ is an ordered pair $[x, y]$, with $x, y \in D$ and $x < y$ (*strict semantics*). We denote by $\mathbb{I}(\mathbb{D})$ the set of all intervals over $\mathbb{D}$. Formulas of RPNL are obtained from a countable set $\mathcal{AP}$ of proposition letters by the following abstract syntax:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle A \rangle\varphi \mid [A]\varphi$$

The other classical connectives can be defined as shorthands.

Formulas are interpreted on models $M = \langle \mathbb{D}, V \rangle$, where $\mathbb{D}$ is a finite linear order and $V : \mathbb{I}(\mathbb{D}) \to 2^{\mathcal{AP}}$ is a *valuation* function that associates every interval of $\mathbb{D}$ with the set of proposition letters that hold true on it. The satisfiability relation $\Vdash$ is defined by the semantic clauses for propositional logic plus the following ones:

– $M, [x, y] \Vdash \langle A \rangle \varphi$ iff $M, [y, z] \Vdash \varphi$, for some z such that $z > y$, and
– $M, [x, y] \Vdash [A]\varphi$ iff $M, [y, z] \Vdash \varphi$, for every z such that $z > y$.

It can be easily checked that $[A]$ is the dual of $\langle A \rangle$, that is, $[A]\varphi$ is equivalent to $\neg\langle A \rangle\neg\varphi$, and vice versa. As shown in [8], satisfiability of RPNL-formulas can be reduced to *initial* satisfiability, that is, satisfiability on the interval $[0, 1]$. Hence, it holds that an RPNL-formula φ is satisfiable if and only if there is a model M such that $M, [0, 1] \Vdash \varphi$.

The decidability proof given in [8] shows that any RPNL-formula φ is satisfiable over finite linear orders if and only if it is satisfiable over a finite linear order whose domain has cardinality strictly less than $2^m \cdot m + 1$, where m is the number of diamonds and boxes in φ. This provides a termination condition that can be used to implement a *fair* decision procedure that exhaustively searches for a model of size smaller than or equal to the bound. In this paper, we develop and implement a tableau-based decision procedure for RPNL by tailoring the general algorithm described in [11] to it and making use of the bound on the size of the model to guarantee completeness.

3 A tableau system for RPNL

The abstract structure of a tableau for an RPNL formula φ is a *rooted tree* where each node is labeled with an *annotated formula* of the form $\theta : [x, y]$, which states that ψ holds over the interval $[x, y]$ (on a finite domain D). A path from the root to a leaf is called a *branch*. Every branch B of the tableau is associated with a finite domain $D_B = \{x_0, x_1, \ldots, x_N\}$ and it represents a partial model for the input formula φ.

At each step of the tableau construction, a branch B and a node n in it are selected, and B is expanded by applying one of the *expansion rules*, which is chosen on the basis of the formula in the label of n.

Before building the tableau, the formula φ to be checked for satisfiability is transformed into an equivalent formula in negated normal form $nnf(\varphi)$, that is, a formula where negation can only occur immediately before proposition letters. Since such a transformation does not change the number of diamonds and boxes, it does not affect the bound on the maximum cardinality of the domain. For such a reason, from now on we will assume that all the formulas are in negated normal form, and thus we do not need to define an expansion rule for negation.

Let B be the selected branch and n, labeled with $\theta : [x_i, x_j]$, be the selected node on B. Expansion rules directly follow the semantics of RPNL (Fig. 1).

The rule (*or*), for disjunctions (θ is of the form $\psi \vee \tau$), expands the branch by adding two disjoint children, each of which is labeled with one of the disjuncts. Two new branches are thus created, which represent the two possible way to satisfy a disjunction, that is, by satisfying either the first or the second disjunct.

The rule (and), for conjunctions (θ is of the form $\psi \wedge \tau$), expands the branch by first adding a child n', labeled with ψ, and then by adding to n' a child n'' labeled with τ. Thus, only one new branch is created, which represents the only possible way to satisfy a conjunction, that is, by satisfying both its conjuncts.

The rule *(dia)*, for existentially quantified formulas (θ is of the form $\langle A\rangle\psi$), explores all possible ways of satisfying the formula ψ (and, for each of them, it generates a new branch): either it satisfies ψ on an existing interval $[x_j, x_h]$, for some $x_h \in D$, with $x_j < x_h$, or a new point x'_h, with $x_j < x'_h$, is added to the domain and ψ is satisfied on the new interval $[x_j, x'_h]$. In the former case, a new branch is generated by adding a node labeled with $\psi : [x_j, x_h]$ to the current branch; in the latter case, the new branch is obtained by adding a node labeled with $\psi : [x_j, x'_h]$ the current branch (see Fig. 1).

Finally, the rule *(box)*, for universally quantified formulas (θ is of the form $[A]\psi$), states that ψ must be true on every existing interval starting at x_j: it appends to the branch a node labeled with $\psi : [x_j, x_h]$, for each interval $[x_j, x_h]$ beginning at x_j.

$$(or)\ \frac{\psi \vee \tau : [x_i, x_j]}{\psi : [x_i, x_j] \mid \tau : [x_i, x_j]} \qquad (and)\ \frac{\psi \wedge \tau : [x_i, x_j]}{\psi : [x_i, x_j], \tau : [x_i, x_j]}$$

$$(dia)\ \frac{\langle A\rangle\psi : [x_i, x_j]}{\psi : [x_j, x_{j+1}] \mid \ldots \mid \psi : [x_j, x_N] \mid \psi : [x_j, x'_j] \mid \ldots \mid \psi : [x_j, x'_N]}$$

$$(box)\ \frac{[A]\psi : [x_i, x_j]}{\psi : [x_j, x_{j+1}], \ldots, \psi : [x_j, x_N]}$$

For each $j \leq h \leq N$, x_h is a point in D_B and x'_h is a new point added to D_B and placed immediately after x_h and, when $h < N$, immediately before x_{h+1}.

Fig. 1: Expansion rules.

It is worth noticing that point x_i never occurs in the consequent of the rules, meaning that the left endpoint of the interval does not play a role in the expansion.

A branch B in the tableau is declared *closed* if either both $p : [x_h, x_k]$ and $\neg p : [x_h, x_k]$, for some $p \in \mathcal{AP}$ and interval $[x_h, x_k]$, occur on it (*contradictory branch*) or the cardinality of the domain D_B is greater than the upper bound on the size of models to be checked for satisfiability given by the small model theorem (*too long branch*). Otherwise, B is considered *open*. Expansion rules are applied only to open branches (closed branches are discarded).

Given a branch B, an annotated formula $\theta : [x_i, x_j]$ is *inactive on B* if:

- θ is a literal;
- θ is of the form $\psi \vee \tau$, $\psi \wedge \tau$, or $\langle A\rangle\psi$ and the rule for θ has been already applied to it on B;

– θ is of the form $[A]\psi$ and for each $x_h > x_j$ there exists a node on B labeled with $\psi : [x_j, x_h]$.

The annotated formula $\theta : [x_i, x_j]$ is *active on B* otherwise.

The branch-expansion strategy applied by the system is the simplest possible one: the first (top-down) active formula of the current branch is selected, expanded, and deactivated[5]. Notice that if the application of the rule *dia* introduces a new point in the domain, the rule *box* is reactivated. Whenever an open branch with no active formulas is found, the procedure terminates with success (the formula is satisfiable). If all branches are closed, the procedure terminates with failure (the formula is unsatisfiable).

As an example of the application of the proposed tableau method, Fig. 2 shows a tableau for the formula $\varphi = \langle A\rangle p \wedge [A](\neg p \vee \langle A\rangle p)$.

Since the formula φ is satisfiable, the tableau contains some open branches, each of which defines (at least) one truth assignment for each proposition letter over each interval. In fact, there may be some interval $[x_i, x_j]$ and some proposition letter $p \in \mathcal{AP}$ such that the truth value of p over $[x_i, x_j]$ is not specified. In such a case, both truth values for p over $[x_i, x_j]$ are admissible. As an effect of this underspecification, more than one model for φ can be, in general, associated with the same open branch.

The left-most branch is closed as it is contradictory (it contains a node labeled with $p : [1, 2]$ and a node labeled with $\neg p : [1, 2]$). The right-most branch adds a new element in the domain every three nodes, and thus will be closed when the size of the domain exceeds the domain upper bound $2^m \cdot m$. All the other branches are open.

The following theorems, which directly follow from the results in [8,11], prove the correctness of the proposed method.

Theorem 1 (termination). *The proposed tableau method terminates.*

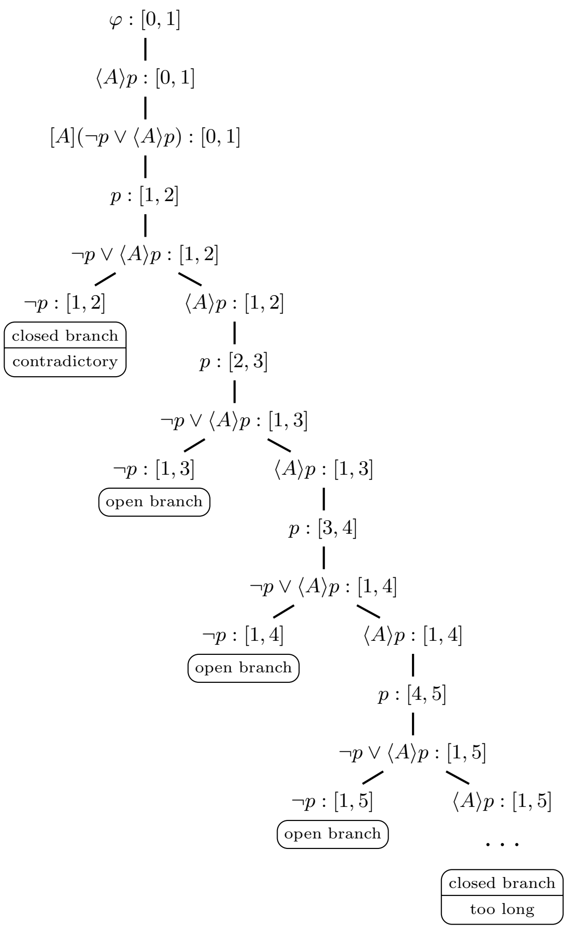
Theorem 2 (soundness). *Let φ be an* RPNL*-formula and T a tableau for φ. If T contains an open branch, then φ is satisfiable.*

Theorem 3 (completeness). *Let φ be an* RPNL*-formula. If φ is satisfiable, then there exists an open branch in every tableau for φ.*

4 Implementation of the tableau system for RPNL

In this section, we illustrate the difficulties we encountered and the implementation choices we made to turn the tableau system described in Section 3 into a computer program. The code of our implementation is written in C++ and it makes no use of external libraries, except for the C++ Standard Library.

[5] The identification of the current branch among all open branches depends on the adopted priority policy. In Section 4, we will explore a number of alternative priority policies.



$\varphi : [0,1]$
$\langle A\rangle p : [0,1]$
$[A](\neg p \vee \langle A\rangle p) : [0,1]$
$p : [1,2]$
$\neg p \vee \langle A\rangle p : [1,2]$
$\neg p : [1,2]$
closed branch
contradictory
$\langle A\rangle p : [1,2]$
$p : [2,3]$
$\neg p \vee \langle A\rangle p : [1,3]$
$\neg p : [1,3]$
open branch
$\langle A\rangle p : [1,3]$
$p : [3,4]$
$\neg p \vee \langle A\rangle p : [1,4]$
$\neg p : [1,4]$
open branch
$\langle A\rangle p : [1,4]$
$p : [4,5]$
$\neg p \vee \langle A\rangle p : [1,5]$
$\neg p : [1,5]$
open branch
$\langle A\rangle p : [1,5]$
...
closed branch
too long

Fig. 2: A tableau for $\varphi = \langle A\rangle p \wedge [A](\neg p \vee \langle A\rangle p)$.

There are two reasons behind our choice. On the one hand, C++ is an high-level language designed for general-purpose programming and that guarantees very good performances in terms of execution speed. On the other hand, the authors were quite familiar with such a language, which means that no time overhead was needed to learn a new programming language. We exploited suitable data structures to represent formulas, nodes, and branches of the tableau, and we developed a search procedure that keeps track of currently-open branches and expands them by applying expansion rules according to the expansion strategy.

Representation of formulas, nodes, and branches

Since in most applications the input formula φ encodes a set of requirements to be jointly satisfied, e.g., those of a plan, we assume φ to be a logical conjunction, whose conjuncts are entered as distinct lines of a text file. Then, φ is stored as a *syntactic tree*, whose leaves are labeled with proposition letters and whose internal nodes are labeled with Boolean connectives and modalities. In such a way, each sub-formula of φ corresponds to a subtree of the syntactic tree. Nodes of the tableau are represented by a structure with four components: a pointer to the subtree representing the formula labeling the node, two integer variables x and y, that identify the interval annotating the formula, and a Boolean flag, which specifies whether the node is active or not. A branch B is implemented as a list of nodes, enriched with two integer variables N and A representing respectively the cardinality of the domain D_B and the number of active nodes.

The search procedure

The search procedure stores the open branches to be expanded into a *priority queue*. At the beginning, the queue contains only the single node initial branch $\{\varphi : [0, 1]\}$. Then, the procedure operates as follows:

1. it extracts the branch B with the highest priority from the queue;
2. it checks whether B is closed or open; if B is closed, then it deletes the branch and it restarts from 1;
3. it finds the closest-to-the-root active node ν in B; if there are no active nodes in B, it terminates with success and it returns B as a model for φ;
4. it applies the appropriate expansion rule to ν, it deactivates ν, it inserts the branches created by the rule into the queue, and it restarts from 1.

The expansion loop is repeated until either a model for φ is found or the queue becomes empty. In the latter case, no model for φ can be found, and the formula is declared unsatisfiable.

Priority policies

The priority policy of the queue determines the next branch to expand. We implemented five different policies:

i) the standard *FIFO* (First In, First Out) policy;
ii) expand the branches with the *smallest domain* first (SDF);
iii) expand the branches with the *largest domain* first (LDF);
iv) expand the branches with the *smallest number of active nodes* first (SAN);
v) expand the branches with the *greatest number of active nodes* first (GAN).

By default, the queue follows the FIFO policy, but the user can easily opt for a different one for a particular problem.

Notice that all the policies are complete: eventually, every branch will be processed and the queue will be empty. The intuitive argument behind this claim is based on the observation that the application of an expansion rule different from (*box*) to an active formula φ on a branch B may cause the generation of a finite number of new branches, where φ is deactivated and will never become active again. Moreover, the labels of the new nodes appended to the branch only contain sub-formulas of φ. Thus, every formula is used only once for a branch expansion and the *overall complexity degree* of the active formulas of the branch (i.e., the sum of the sizes of the active formulas of the branch) decreases at every expansion, unless the branch is expanded by means of a rule (*box*). However, even if a rule (*box*) can be applied several time, it will not be applied infinitely many times. To convince oneself that this is true, notice that, after the first application and subsequent deactivation, the rule (*box*) can become active only after the insertion of a new element in the domain. Since domains whose size exceeds the upper bound $2^m \cdot m$ are closed and discarded, every rule (*box*) is applied finitely many times only. Thus, since the overall complexity degree of a branch will eventually decrease to 0, and since only finitely many branches are created at each step, the thesis is a direct consequence of the König's Lemma.

Branch expansion strategy

If the current branch B (extracted from the queue) is declared open at step 2 of the search procedure, nodes in B are scanned to determine the closest-to-the-root active node ν. The expansion of B depends on the shape of the formula labeling ν. Three cases are possible.

Boolean formula. Since formulas are assumed to be in negated normal form, the only possible rules are the $\vee$-rule and the $\wedge$-rule. Let ν be labeled with $\psi \vee \tau : [x_i, x_j]$ (the case $\psi \wedge \tau : [x_i, x_j]$ is similar and thus omitted). We must distinguish four scenarios:

(i) both $\psi : [x_i, x_j]$ and $\tau : [x_i, x_j]$ are already on B,
(ii) $\psi : [x_i, x_j]$ is on B, while $\tau : [x_i, x_j]$ is not,
(iii) $\tau : [x_i, x_j]$ is on B, while $\psi : [x_i, x_j]$ is not, and
(iv) neither of the two is on B.

In case (i), there is no need to apply the rule: $\psi \vee \tau : [x_i, x_j]$ is deactivated and B is put back in the queue. In case (ii), a copy of the branch is generated and the annotated formula $\tau : [x_i, x_j]$ is added to it; then, $\psi \vee \tau : [x_i, x_j]$ is deactivated and both B and its copy are added to the queue. Case (iii) is completely symmetric. In case (iv), two copies of the branch are generated: one is expanded with the annotated formula $\psi : [x_i, x_j]$, the other one with $\tau : [x_i, x_j]$. Then, $\psi \vee \tau : [x_i, x_j]$ is deactivated, both copies of B are added to the queue, and the original B is discarded.

Box formula $[A]\psi : [x_i, x_j]$. The box rule is applied. First, we deactivate the formula $[A]\psi : [x_i, x_j]$; then, for each $x_j < x_h \leq x_N$, if $\psi : [x_j, x_h]$ does not belong to B, we add it; finally, the expansion of B is inserted into the queue.

Diamond formula $\langle A\rangle\psi : [x_i, x_j]$. The diamond rule is applied. First, we check whether for some $x_h > x_j$ the annotated formula $\psi : [x_j, x_h]$ is on B. If this is the case, we deactivate $\langle A\rangle\psi : [x_i, x_j]$ and we put B back in the queue. Otherwise, we create a distinct copy of B for every possible way of satisfying ψ:

– $N-j$ copies $B_{j+1}, \ldots, B_N$, with domain cardinality N, that will be expanded with the annotated formulas $\psi : [x_j, x_{j+1}], \ldots, \psi : [x_j, x_N]$, respectively;
– $N-j+1$ copies $B'_j, \ldots, B'_N$, with domain cardinality $N+1$, that will be expanded with the annotated formulas $\psi : [x_j, x'_j], \ldots, \psi : [x_j, x'_N]$, respectively.

For each copy B'_h, the expansion of the domain is obtained as follows:

(i) every annotated formula $\tau : [x_k, x_l]$ such that $x_k > x_h$ is replaced by the annotated formula $\tau : [x_k + 1, x_l + 1]$. If $x_k \leq x_h < x_l$, the annotated formula is replaced by $\tau : [x_k, x_l + 1]$, while if $x_l \leq x_h$, the annotated formula remains unchanged;
(ii) we add a new node labeled with the annotated formula $\psi : [x_j, x_h + 1]$;
(iii) we reactivate all annotated formulas $[A]\tau : [x_k, x_l]$ with $x_l \leq x_h$.

To conclude the expansion, we deactivate $\langle A\rangle\psi : [x_i, x_j]$, we put all $2 \cdot (N-j)+1$ copies of B in the queue, and we discard B.

Usage

The implementation described here has been made accessible at the web page `http://www.di.unisa.it/dottorandi/dario.dellamonica/tableaux/`. By means of a user-friendly web interface it is possible to check the satisfiability of RPNL-formulas over finite linear orders. Formulas are provided in input by means of plain-text files in the following format:

– the first line must contain the integer N, denoting the number of different proposition letters occurring in the formula, followed by the list of such letters, that must have the form `p0 p1 p2 ... p<N-2> p<N-1>`;
– the second line must contain the integer M, representing the number of lines in the file the formula spreads out;
– each of the next M lines must contain an RPNL-formula: the program will read such lines and will treat each line as a conjunct of the input formula. Formulas must follow the following grammar:

f ::= p | `NOT ( ` f ` )` | `( ` f ` OR ` f ` )` | `( ` f ` AND ` f ` )` | `DIA ( ` f ` )` | `BOX ( ` f ` )`
p ::= `p0` | `p1` | `p2` | `...` | `p<N-2>` | `p<N-1>`

A zip archive containing the problems used for the tests (see the next section) is available at the above-mentioned web page. In Fig. 3, a sample input file is shown.

In the main screen the user is asked to input the input file containing the RPNL-formula to be checked for satisfiability. By clicking on the button "Next" a new page is shown, where the user can select the following parameters:

```
4 p0 p1 p2 p3
10
DIA ( p0 )
DIA ( p1 )
DIA ( p2 )
DIA ( p3 )
BOX ( NOT ( ( p0 AND p1 ) ) )
BOX ( NOT ( ( p0 AND p2 ) ) )
BOX ( NOT ( ( p0 AND p3 ) ) )
BOX ( NOT ( ( p1 AND p2 ) ) )
BOX ( NOT ( ( p1 AND p3 ) ) )
BOX ( NOT ( ( p2 AND p3 ) ) )
```

Fig. 3: Example of input file.

- the number of branches in the tableau under which the policy FIFO is applied by default, thus ignoring a possible different choice made by the user (see below);
- the maximum size of the domain: branches whose domain size exceeds the value of this parameter are closed and discarded. The default value is the upper bound $2^m \cdot m$ (where m is the number of modalities in the input formula), which guarantees the completeness of the procedure. Anyway, the user can select a different (smaller) value to improve the performance, at risk of losing the completeness;
- the priority policy (see above): 0 for the FIFO policy (this is the default value), 1 for the GAN policy, 2 for the SAN policy, 3 for the LDF policy, and 4 for the SDF policy;
- finally, it is possible to choose to get a fully-detailed output, that means that all the steps of the tableau construction will be displayed.

5 Experiments

We have tested our implementation against a benchmark of different problems, divided into two classes. First, we tested the scalability of the program with respect to a set of combinatorial problems of increasing complexity (COMBINATORICS), where the n-th combinatorial problem is defined as the problem of finding a model for the formula that contains n conjuncts, each one of the type $\langle A\rangle p_i$ $(0 \leq i \leq n)$, plus $\frac{n(n+1)}{2}$ formulas of the type $[A]\neg(p_i \wedge p_j)$ $(i \neq j)$. Then, we considered the set of 36 "easy" purely randomized formulas used in [5] to evaluate an Evolutionary Computation algorithm for RPNL finite satisfiability (RANDOMIZED). Table 1 summarizes the outcome of our experiments. For each class of problems, the corresponding table shows, for each instance n, the time necessary to solve the problem for each policy (FIFO, SDF, LDF, SAN, GAN) and the size of the obtained model (or "unsat" if the instance was proved

COMBINATORICS

	Policy (sec)					Outcome
n	**FIFO**	**SDF**	**LDF**	**SAN**	**GAN**	**(size)**
1	0.004	0.004	0.004	0.004	0.004	4
2	0.004	0.008	0.004	0.004	0.008	5
3	0.008	0.15	0.03	0.008	0.03	6
4	0.01	–	30.07	0.01	30.29	7
5	0.012	–	–	0.012	–	8
6	0.02	–	–	0.03	–	9
7	0.07	–	–	0.07	–	10
8	0.15	–	–	0.16	–	11
9	0.3	–	–	0.32	–	12
10	0.56	–	–	0.59	–	13
11	0.99	–	–	1.06	–	14
12	1.67	–	–	1.79	–	15
13	2.73	–	–	2.94	–	16
14	4.25	–	–	4.55	–	17
15	6.56	–	–	7.08	–	18
16	9.77	–	–	10.82	–	19
17	14.42	–	–	15.40	–	20
18	20.79	–	–	22.20	–	21
19	29.28	–	–	32.11	–	22
20	40.91	–	–	44.09	–	23
21	–	–	–	–	–	–
22	–	–	–	–	–	–

RANDOMIZED

	Policy (sec)					Outcome
n	**FIFO**	**SDF**	**LDF**	**SAN**	**GAN**	**(size)**
1	0.004	0.004	0.004	0.004	0.004	4
2	0.004	0.004	0.004	0.004	0.004	4
3	0.004	0.004	0.004	0.004	0.004	4
4	0.004	0.004	0.004	0.004	0.004	4
5	0.004	0.004	0.004	0.004	0.004	4
6	0.004	0.004	0.004	0.004	0.004	4
7	0.07	0.23	0.004	0.18	0.004	3 / 4
8	0.004	0.004	0.004	0.004	0.004	4
9	0.004	0.004	0.004	0.004	0.004	4
10	0.004	0.004	0.004	0.004	0.004	4
11	0.004	0.004	0.004	0.004	0.004	4
12	0.004	0.004	0.004	0.004	0.004	4
13	0.01	0.04	0.004	0.02	0.004	4
14	0.004	0.004	0.004	0.004	0.004	4
15	0.004	0.004	0.004	0.004	0.004	4
16	0.004	1.37	0.004	0.01	0.004	4
17	0.004	0.004	0.004	0.004	0.004	4
18	0.004	0.004	0.004	0.004	0.004	3
19	1.66	45.43	0.68	1.91	0.02	3 / 4
20	0.02	0.004	0.03	0.03	0.004	2 / 4
21	0.004	0.004	0.004	0.004	0.004	4
22	0.74	14.08	0.004	1.04	0.004	4
23	0.004	0.004	0.004	0.004	0.004	4
24	0.004	0.004	0.004	0.004	0.004	4
25	–	–	–	–	–	–
26	0.004	0.004	0.004	0.004	0.004	4
27	0.004	–	0.004	0.01	–	3 / 4
28	0.004	0.004	0.004	0.004	0.004	4
29	0.004	–	0.004	0.004	0.004	4
30	0.14	0.08	0.04	0.19	0.01	2 / 4
31	0.004	0.004	0.004	0.004	0.004	unsat
32	0.25	–	0.02	0.31	0.004	2 / 4
33	0.004	0.004	0.004	0.004	0.004	4
34	–	–	0.02	0.004	0.02	2 / 4
35	0.004	–	0.004	–	0.004	2 / 4
36	–	–	–	–	1.2	3

Table 1: Experimental results.

to be unsatisfiable). A time-out of 1 minute was used to stop instances running for too long. All the experiments were executed on a notebook with an Intel Pentium Dual-Core Mobile 1.6 Ghz CPU and 2 Gb of RAM, under Ubuntu Linux 11.04.

Despite being a prototypical implementation, our system runs reasonably well on the COMBINATORICS benchmark, being able to produce a result in a short time for formulas up to 20 conjuncts (and up to a model size of 23 points). The results of the RANDOMIZED benchmark allows for a first comparison with the Evolutionary algorithm in [5], and shows that the two algorithms have similar performances on the considered formulas. The tableau system was able to prove that problem 31 is unsatisfiable, while the evolutionary algorithm (being incomplete) can only provide positive answers.

6 Conclusions and future work

In this paper, we described a first working implementation of a decision procedure for the logic of Allen's relation *meets*, a.k.a. Right Propositional Neighborhood Logic (RPNL), interpreted over finite linear orders. The proposed implementation of the tool, which has been made accessible at the web page `http://www.di.unisa.it/dottorandi/dario.dellamonica/tableaux/`, is based on the original (non-terminating) tableau system given in [11] and it exploits the small model theorem proved in [8] to guarantee termination.

At the best of our knowledge, there are no available benchmarks for RPNL or any other interval temporal logic. Thus, the procedure has been tested against (suitably adapted) benchmarks relative to different problems. In particular, we were interested in testing the scalability of our software tool and in comparing its efficiency with that of an existing Evolutionary Computation algorithm for checking finite satisfiability of RPNL formulas [5]. The outcomes of such a testing are presented and, despite being a prototypical implementation, they show that the developed system runs reasonably well on the considered benchmark.

As for future work, we have set ourself to the goal of adapting some benchmarks for the modal logic K [2] and the temporal logic LTL [17] to the interval semantics to obtain a larger and more conclusive benchmark for RPNL. We are also thinking of extending the current implementation along two different directions. On the one hand, we are interested in developing a tool for deciding RPNL over a number of classes of linear orders including (but not limited to) the natural numbers, the class of dense linear orders, and the class of all linear orders; on the other hand, we plan to generalize the decision procedure for RPNL to full Propositional Neighborhood Logic (PNL), which adds a modality for the Allen relation *met-by* (the inverse of the Allen relation *meets*) to RPNL [7].

Acknowledgments

The authors acknowledge the support from the Spanish fellowship program *'Ramon y Cajal' RYC-2011-07821* and the Spanish MEC project *TIN2009-14372-C03-01* (G. Sciavicco), the project *Processes and Modal Logics* (project nr. 100048021) of the Icelandic Research Fund and the project *Decidability and Expressiveness for Interval Temporal Logics* (project nr. 130802-051) of the Icelandic Research Fund in partnership with the European Commission Framework 7 Programme (People) under "Marie Curie Actions" (D. Della Monica), and the Italian GNCS project *Logiche di Gioco estese* (D. Bresolin and A. Montanari).

References

1. J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
2. P. Balsiger, A. Heuerding, and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. *J. of Automated Reasoning*, 24(3):297–317, 2000.

3. D. Bresolin, V. Goranko, A. Montanari, and P. Sala. Tableaux for logics of subinterval structures over dense orderings. *J. of Logic and Computation*, 20(1):133–166, 2010.
4. D. Bresolin, V. Goranko, A. Montanari, and G. Sciavicco. Propositional interval neighborhood logics: Expressiveness, decidability, and undecidable extensions. *Annals of Pure and Applied Logic*, 161(3):289–304, 2009.
5. D. Bresolin, F. Jiménez, G. Sánchez, and G. Sciavicco. Finite satisfiability of propositional interval logic formulas with multi-objective evolutionary algorithms. In *Proc. of the 12th Workshop on FOundations of Genetic Algorithms (FOGA)*, pages 25–36, 2013.
6. D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Interval temporal logics over finite linear orders: the complete picture. In *Proc. of the 20th European Conference on Artificial Intelligence (ECAI)*, pages 199–204, 2012.
7. D. Bresolin, A. Montanari, P. Sala, and G. Sciavicco. Optimal tableau systems for propositional neighborhood logic over all, dense, and discrete linear orders. In *Proc. of the 20th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*, volume 6793 of *LNAI*, pages 73–87. Springer, 2011.
8. D. Bresolin, A. Montanari, and G. Sciavicco. An optimal decision procedure for Right Propositional Neighborhood Logic. *J. of Automated Reasoning*, 38(1-3):173–199, 2007.
9. D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. Interval temporal logics: a journey. *Bulletin of the European Association for Theoretical Computer Science*, 105:73–99, 2011.
10. L. Fariñas del Cerro, D. Fauthoux, O. Gasquet, A. Herzig, D. Longin, and F. Massacci. Lotrec: the generic tableau prover for modal and description logics. In *Proc. of the 1st International Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *LNCS*, pages 453–458. Springer, 2001.
11. V. Goranko, A. Montanari, and G. Sciavicco. Propositional interval neighborhood temporal logics. *J. of Universal Computer Science*, 9(9):1137–1167, 2003.
12. V. Goranko, A. Montanari, and G. Sciavicco. A road map of interval temporal logics and duration calculi. *J. of Applied Non-Classical Logics*, 14(1–2):9–54, 2004.
13. J. Halpern and Y. Shoham. A propositional modal logic of time intervals. *J. of the ACM*, 38(4):935–962, 1991.
14. H. Kamp and U. Reyle. *From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory, Volume 42 of Studies in Linguistics and Philosophy.* Springer, 1993.
15. B. Moszkowski. *Reasoning about digital circuits.* Tech. rep. stan-cs-83-970, Dept. of Computer Science, Stanford University, Stanford, CA, 1983.
16. I. Pratt-Hartmann. Temporal prepositions and their logic. *Artificial Intelligence*, 166(1-2):1–36, 2005.
17. K. Y. Rozier and M. Y. Vardi. LTL satisfiability checking. *Int. J. on Software Tools for Technology Transfer*, 2(12):123–137, 2010.