# Reasoning about general TBoxes with spatial and temporal constraints: Implementation and optimizations

Matthias Hengel, Stefan Wölfl, and Bernhard Nebel

Department of Computer Science, University of Freiburg,
Georges-Köhler-Allee 52, 79110 Freiburg, Germany
{hengelm,woelfl,nebel}@informatik.uni-freiburg.de

**Abstract.** In many applications a reasonable representation of conceptual knowledge requires the possibility to express spatial or temporal relations between domain entities. A feasible approach to this is to consider spatial or temporal constraint systems on concrete domains. Indeed, Lutz and Miličić (2007) showed that for the description logic $\mathcal{ALC}(\mathcal{C})$ with $\omega$-admissible constraint systems, concept satsifiability with respect to general TBoxes can be decided by an extension of a standard $\mathcal{ALC}$ tableau algorithm. In this paper we report on a study in which this tableau method was implemented, optimized, and evaluated.

## 1 Introduction

When an application domain is to be conceptualized in terms of a terminological knowledge base, it is often necessary to refer to spatial or temporal relations between domain entities. Examples include domains dealing with legal concepts (such as traffic laws, building legislation, etc.), architectural domains, and domains concerned with spatial layouts. At first glance it seems suggesting to use spatial and temporal relations as roles in description logic TBoxes, but a series of undecidability results (see, e.g., [21]) lead to a different approach in which such relations are handled as constraints on concrete spatial or temporal domains (see, e.g., [2, 10]). This concrete domain approach may result in undecidable reasoning problems (in particular if one allows for general concept inclusions, see, e.g., [10]), but Lutz and Miličić [11] showed that so-called $\omega$-admissible constraint systems on concrete domains allow for deciding consistency of general TBoxes in the description logic $\mathcal{ALC}(\mathcal{C})$. In particular, they presented a tableau algorithm and stated that the algorithm is a first step towards an efficient implementation of description logics with constraint systems on concrete domains.

In this paper we present the first implementation of this tableau procedure.[1] In particular, we are interested in the question which of the standard optimization techniques used in current description logic reasoners [7] can be used in a reasoner for $\mathcal{ALC}(\mathcal{C})$. We show that techniques such as lazy unfolding and absorption can be effortlessly applied

[1] Note that serveral DL-reasoners support concrete domain reasoning (see, e.g., [19, 17, 16], but to our knowledge, there is currently no reasoner that supports reasoning with *constraint systems* on concrete domains. Furthermore there has been some work on integrating spatial reasoning into description logics using a hybrid approach [5, 18].

within a concrete domain DL-reasoner, while other techniques such as dependency-directed backtracking and caching need careful adaptations.

A second source for optimizations are special techniques that arise from interweaving tableau reasoning and constraint-based reasoning. For example, spatial and temporal constraint-based formalisms such as RCC8 [13] and Allen's interval algebra [1] typically allow for disjunctive constraints. Since such constraints can be handled by constraint solvers for these languages, it makes sense to delegate reasoning steps.

The setup of the paper is as follows: In section 2, we provide the necessary background on the description logic $\mathcal{ALC}(\mathcal{C})$ as well as the tableau procedure for $\mathcal{ALC}(\mathcal{C})$ as presented in [11]. In section 3, we discuss which standard tableau optimization techniques can be adapted to the $\mathcal{ALC}(\mathcal{C})$-tableau procedure and also the above mentioned special techniques that improve the processing of disjunctive constraints. In section 4, we report on the results of an empirical evaluation of our implementation of the $\mathcal{ALC}(\mathcal{C})$-tableau procedure, showing the influence of the different optimization techniques. Since there are no knowledge bases yet that use this particular language, we have to generate synthetic test cases. However, the structure of our test cases tries to mimic the structure of existing KBs. Section 5 concludes the paper.

## 2 Background

In what follows we introduce the technical background in a rather condensed way. For a more detailed presentation we refer to [11]. Furthermore we assume that the reader is familiar with the description logic $\mathcal{ALC}$ [15].

**Constraint systems.** A *constraint system* on a concrete domain $D$ is a finite set $\Gamma$ of binary relations on $D$. Usually the relations in the constraint system are assumed to be jointly exhaustive and pairwise disjoint. Given a constraint system $\Gamma$, its *disjunctive closure* $\Gamma^\vee$ is the set of all relations that are unions of relations in $\Gamma$. We write relations from $\Gamma^\vee$ in the form $r_1 \vee \cdots \vee r_n$ with $r_i \in \Gamma$ $(1 \leq i \leq n)$. A *constraint*, then, is an expression of the form $(v\,r\,v')$ where $r$ is a relation from the disjunctive closure of $\Gamma$ and $v$ and $v'$ are variables from a countably infinite set of variables. A *constraint network* $N$ is a finite or infinite set of constraints (we use the symbol $V_N$ to denote to the set of all variables mentioned in $N$). Given network $N$ and some subset $V' \subseteq V_N$, $N|_{V'}$ is the set of all constraints in $N$ that mention variables from $V'$ only. We will assume that networks contain at most one constraint $(v\,r\,v')$ for each pair of variables $v, v'$. If a network contains exactly one constraint for each variable pair, it is called *complete*. A network is called *atomic* if all its constraints use only relations from the basic constraint system (i.e., it uses no relation from $\Gamma^\vee \setminus \Gamma$). A network $N$ is called *satisfiable* if there exists a mapping $a$ that assigns to each variable in $V_N$ an element of the concrete domain such that for each constraint $(v\,r\,v')$ in $N$, it holds $(a(v), a(v')) \in r$. The variable assignment $a$ is then called a *solution* of the network. A network $N$ is called a *refinement* of network $N'$ if for each constraint $(v\,r\,v')$ in $N$, if $N'$ contains a constraint of the form $(v\,r'\,v')$ then $r \subseteq r'$ (i.e., each relation $r_j \in \Gamma$ mentioned in $r$ is also mentioned in $r'$).

Examples of constraint systems include Allen's interval calculus [1], which consists of 13 relations describing all possibilities how in a linear flow of time the start- and endpoints of two intervals can be related to each other. Another example is the fragment RCC8 of the Region Connection Calculus [13] with 8 relations describing topological relations between extended spatial regions. For further examples as well as for basic algorithms used to decide the satisfiability of such constraint networks, we refer to [14].

A constraint system is said to have the *compactness property* if for each infinite network $N$, $N$ is satisfiable if and only if for each finite subset $V' \subseteq V_N$ the network $N|_{V'}$ is satisfiable. It is said to have the *patchwork property* if for any pair of finite, atomic, complete, and satisfiable networks $N$ and $N'$ that coincide on all constraints with variables from $V_N \cap V_{N'}$, the network $N \cup N'$ is satisfiable. Finally, the constraint system is called *$\omega$-admissible* if it has both the compactness and the patchwork property and furthermore, there exists a procedure that decides the satisfiability of finite networks. Note that both Allen's interval calculus and RCC8 feature $\omega$-admissibility [11].

**$\mathcal{ALC}(\mathcal{C})$.** In what follows let $\mathcal{C} = \langle D, \Gamma \rangle$ be a fixed $\omega$-admissible constraint system. The vocabulary of the description logic $\mathcal{ALC}(\mathcal{C})$ is given by countably infinite sets of *concept names* $(A, A', A_1, \ldots)$, *role names* $(R, R', R_1, \ldots)$ with an infinite subset of *abstract features*, and *concrete features* $(g, g', g_1, \ldots)$. A *path* is a sequence $R_1 \ldots R_k g$ consisting of role names $R_i$ and a single concrete feature $g$. A *feature path* is a path $R_1 \ldots R_k g$, where all role names $R_i$ are abstract features. The *concepts* of $\mathcal{ALC}(\mathcal{C})$ are formed by the following rule:

$$C ::= A \mid \neg C \mid C \sqcap C' \mid C \sqcup C' \mid \exists R.C \mid \forall R.C \mid \exists U_1, U_2.r \mid \forall U_1, U_2.r$$

with $r \in \Gamma^{\vee}$. In the *constraint constructors* $\exists U_1, U_2.r$ and $\forall U_1, U_2.r$ we allow the following two situations: (a) both $U_1$ and $U_2$ are feature paths or (b) both have length $\leq 2$, i.e., they have the form $U_i = Rg_i$ or $U_i = g_i$ $(i = 1, 2)$). A *general concept inclusion* is an expression of the form $C \sqsubseteq C'$ where $C$ and $C'$ are arbitrary $\mathcal{ALC}(\mathcal{C})$-concepts. Finally, a *general TBox* is a finite set of general concept inclusions.

We use a descriptive set-based semantics: An *interpretation* $\mathcal{I}$ is a tuple $(\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta_{\mathcal{I}}$ is called the *domain* and $\cdot^{\mathcal{I}}$ the *interpretation function*. The interpretation function maps each abstract feature $f$ to a partial function from $\Delta_{\mathcal{I}}$ to $\Delta_{\mathcal{I}}$, and each concrete feature $g$ to a partial function from $\Delta_{\mathcal{I}}$ to the concrete domain $D$. For pure $\mathcal{ALC}$-concepts we use the ordinary semantics as given in [15]. The interpretation function is then extended to arbitrary $\mathcal{ALC}(\mathcal{C})$-concepts as follows:

$$(\exists U_1, U_2.r)^{\mathcal{I}} := \{x \in \Delta_{\mathcal{I}} \colon \exists d_1 \in U_1^{\mathcal{I}}(x),\ d_2 \in U_2^{\mathcal{I}}(x) \text{ with } \langle D, \Gamma \rangle \models (d_1\, r\, d_2)\}$$
$$(\forall U_1, U_2.r)^{\mathcal{I}} := \{x \in \Delta_{\mathcal{I}} \colon \forall d_1 \in U_1^{\mathcal{I}}(x),\ d_2 \in U_2^{\mathcal{I}}(x), \text{ it holds } \langle D, \Gamma \rangle \models (d_1\, r\, d_2)\},$$

In these conditions we use the following notations: $\langle D, \Gamma \rangle \models (d_1\, r\, d_2)$ means $(d_1, d_2) \in r_i$ for some $1 \leq i \leq n$ (where $r = r_1 \vee \cdots \vee r_n$); and for a path $U = (R_1 \ldots R_k g)$, $U^{\mathcal{I}}(x)$ is defined as:

$$U^{\mathcal{I}}(x) := \{d \in \Delta_{\mathcal{I}} \colon \exists x_1, \ldots, x_{k+1} \in \Delta_{\mathcal{I}}, \text{ such that } x = x_1,\ g^{\mathcal{I}}(x_{k+1}) = d, \\ \text{and } (x_j, x_{j+1}) \in R_j^{\mathcal{I}} \text{ for } 1 \leq j \leq k, \}.$$

**Example 1.** In the language $\mathcal{ALC}(\mathcal{C})$ with the Allen interval constraints, we can for example define the concept of time-travelling father, i.e., a father that is not born before the lifetime of some of his children. Using the role name `has-child`, the concrete feature `has-lifetime`, and the usual notation for interval relations: s ('starts'), d ('during'), f ('finishes'), = ('equals'), > ('after'), mi (converse of 'meets'), oi (converse of 'overlaps'), and si (converse of 'starts'), we can define:

$$\texttt{Timetravelling\_Father} = \texttt{Man} \sqcap$$
$$\exists(\texttt{has-lifetime}),(\texttt{has-child has-lifetime}).(\mathrm{s} \vee \mathrm{d} \vee \mathrm{f} \vee = \vee > \vee \mathrm{mi} \vee \mathrm{oi} \vee \mathrm{si})$$

**Tableau algorithm.** Lutz and Miličič [11] present a tableau algorithm to decide the satisfiability of a concept $C$ w.r.t. a general TBox $\mathcal{T}$ (i.e., the problem if there exists an interpretation that satisfies the concept inclusions in the TBox and assigns to $C$ a non-empty set). The idea of the algorithm is to construct a 'pre-model' step-by-step such that in each step all concrete domain constraints are satisfied. The non-deterministic algorithm builds up a tree with two different kinds of nodes: *abstract* nodes representing individuals in the domain $\Delta_I$ and *concrete* nodes representing variables for concrete domain values. Constraints between the concrete nodes are represented by a constraint network $\mathcal{N}$, which initially is the empty network and which is updated as the algorithm proceeds.

The algorithm starts by creating a single abstract node $a$ labeled with a set containing the concept to be checked (the label set of node $a$ is denoted by $\mathcal{L}(a)$). The label sets are then expanded by applying the tableau rules on the contained concepts. Where needed, abstract successors (nodes corresponding to role successors) and concrete successors (nodes for concrete feature fillers) are created. Successors are labeled as well as the edges linking parents to successors (with role names or concrete features, resp.).

The algorithm expects both the concept to be checked and the TBox to be in *path normal form*, i.e. to be in negation normal form such that every path has length at most two. In [11] it is shown that concept satisfiability w.r.t. general TBoxes can be reduced in polynomial time to satisfiability of concepts in path normal form w.r.t. TBoxes in path normal form.

For pure DL concepts like concept conjunctions, concept disjunctions and role restrictions the standard tableau rules (R$\sqcap$), (R$\sqcup$), (R$\exists$), (R$\forall$) and (RTBox) are used. In addition to these, the algorithm uses the rules listed in Table 1.

To guarantee termination, the algorithm uses a static blocking mechanism that considers the concrete neighborhood of abstract nodes. The *concrete neighborhood* of an abstract node $a$ is defined as the set

$$\mathcal{N}(a) := \{(v\,r\,v') \in \mathcal{N} : \exists g, g' \in \text{feat}(a) \text{ such that } v \text{ } g\text{-successor of } a \text{ and } v' \text{ } g'\text{-successor of } a\},$$

where feat$(a)$ denotes the set of all concrete features $g$ such that $a$ has $g$-successors.

A node $a$ is *potentially blocked* by node $b$ if $b$ is an ancestor of $a$, $\mathcal{L}(a) \subseteq \mathcal{L}(b)$, and feat$(a)$ = feat$(b)$. Node $a$ is *directly blocked* by $b$ if $a$ is potentially blocked by $b$, the concrete neighborhoods $\mathcal{N}(a)$ and $\mathcal{N}(b)$ are atomic and complete, and the mapping

**Table 1.** Subset of the tableau rules for $\mathcal{ALC}(\mathcal{C})$. $a$ and $b$ refer to abstract nodes in the constructed completion system, $\mathcal{N}$ to the constructed constraint network. The presentation here is slightly simplified. For details and a complete list of rules see [11].

| | |
|---|---|
| R$\exists_c$ | If $\exists U_1, U_2.r \in \mathcal{L}(a)$, $r = r_1 \vee \cdots \vee r_n$, $a$ is not blocked, and there exist no variables $v_1, v_2$ in $\mathcal{N}$ such that $v_j$ is a $U_j$-successor of $a$ for $j = 1,2$ and $(v_1\, r_i\, v_2) \in \mathcal{N}$ for some $i$ with $1 \leq i \leq n$, then add 'fresh' $U_1$- and $U_2$-successors $v_1$ and $v_2$, resp., and set $\mathcal{N} := \mathcal{N} \cup \{(v_1\, r_i\, v_2)\}$ for some $i$ with $1 \leq i \leq n$. |
| R$\forall_c$ | If $\forall U_1, U_2.r \in \mathcal{L}(a)$, $r = r_1 \vee \cdots \vee r_n$, $a$ is not blocked, and there are variables $v_1, v_2$ in $\mathcal{N}$ such that $v_j$ is a $U_j$-successor of $a$ for $j = 1,2$ but $(v_1\, r_i\, v_2) \notin \mathcal{N}$ for each $i$ with $1 \leq i \leq n$, then set $\mathcal{N} := \mathcal{N} \cup \{(v_1\, r_i\, v_2)\}$ for some $i$ with $1 \leq i \leq n$. |
| RNet | If $a$ is potentially blocked by $b$ or vice versa, and $\mathcal{N}(a)$ is not complete and atomic, then non-deterministically guess an atomic completion $\mathcal{N}'$ of $\mathcal{N}(a)$ and set $\mathcal{N} := \mathcal{N} \cup \mathcal{N}'$. |

from $V_{\mathcal{N}(a)}$ to $V_{\mathcal{N}(b)}$ induced by assigning to every unique $g$-successor of $a$ the unique $g$-successor of $b$ is an isomorphism. A node $a$ is *blocked* if it or one of its ancestors is directly blocked. On blocked nodes no rule may be applied.

Crucial is the rule (RNet). This rule must be applied before any other rule in order to resolve any potential blocking situation to either a non-blocking or a blocking situation. The rule is used to guess a complete and atomic refinement of the concrete neighborhood of an abstract node $a$.

The non-deterministic algorithm returns unsatisfiable if a 'guessed' completion step yields a *clash*: A clash can arise on the part of DL (e.g. $A$ and $\neg A$ are in a label) or on the part of the concrete domain (the constructed network is unsatisfiable). The algorithm terminates and returns satisfiable if no rule can be applied anymore and no clash is in the guessed completion.

To obtain a model from the 'pre-model' constructed by the algorithm, that part of the 'pre-model' that links the blocked node and its blocking node is 'glued' together infinitely often. In the case of $\mathcal{ALC}(\mathcal{C})$, compactness and the patchwork property are needed to ensure that the so constructed infinite network still is satisfiable. The tableau algorithm runs in 2-NExpTime if satisfiability of the concrete domains is in NP [11]. To the best of our knowledge no tighter complexity results are known.

## 3 Optimizations

Optimizations are essential to gain acceptable runtimes of the tableau algorithm. In [7, 8] several techniques for optimizing satisfiability checking using a tableau algorithm are described. We adapt these techniques, where possible, to our case and additionally propose some techniques which generalize the ideas behind the fragment of $\mathcal{ALC}(\mathcal{C})$ described in [11].

### 3.1 Standard techniques

The idea of *lazy unfolding* is to keep the structure of the knowledge base as long as possible [3]. It works by only introducing axioms which are relevant for the current label. For example, an axiom $\mathrm{A} \sqsubseteq \mathrm{C}$ is only relevant to a label already containing A. It is easy to see that lazy unfolding is applicable in the context of $\mathcal{ALC}(\mathcal{C})$ without restriction: as described for example in [7], general TBoxes may be separated in an unfoldable part, which is handled by lazy unfolding, and a non-unfoldable part, handled by (RTBox).

*Absorption* [3] is used to reduce the number of general concept inclusions in a knowledge base. The idea is to replace general concept inclusions, where possible, by primitive definition axioms, which in a next step may be merged further on. Since this standard technique can be implemented by just considering operations on the label sets of abstract nodes, it is quite clear to see that absorption is applicable to our case without any restriction.

*Backjumping* [8] is a dependency-directed backtracking scheme used to reduce the number of branches expanded during search. It works by identifying the cause behind a clash and using this information to safely ignore branches. Backjumping can be applied to $\mathcal{ALC}$ without any restrictions, but for application to a concrete domain it is necessary to identify the cause of concrete clashes in detail [20]. As our implementation (as described in the next section) uses an external constraint solving library, we perform backjumps only when clashes on the DL part are discovered. When clashes on the part of the constraint reasoner are discovered, we use the chronological backtracking scheme.

*Caching* [8] is a technique that aims at avoiding recomputations. If a node $a$ of a tableau with starting label $\mathcal{L}(a)$ has been discovered to be satisfiable and later in the execution of the algorithm another node $a'$ with $\mathcal{L}(a') = \mathcal{L}(a)$ is created, the idea is to not further expand $a'$ but to reuse the result of the completion of $a$ instead. This works for $\mathcal{ALC}$, as the satisfiability of a node $a$ is independent of the satisfiability of its parent node $b$ once $b$ is completely expanded. But this is not the case in $\mathcal{ALC}(\mathcal{C})$, as the following example shows.

**Example 2.** Let R be an abstract feature (i.e. R is functional) and $\mathcal{T}$ be the following TBox:

$$\mathrm{A} \sqsubseteq \exists \mathrm{R}.(\exists(\mathrm{loc}_1),(\mathrm{loc}_2).(>)) \sqcap \exists(\mathrm{loc}_1),(\mathrm{R\ loc}_1).(=) \sqcap \exists(\mathrm{loc}_2),(\mathrm{R\ loc}_2).(=),$$
$$\mathrm{B} \sqsubseteq \exists(\mathrm{loc}_1),(\mathrm{loc}_2).(<) \sqcap \exists(\mathrm{loc}_1),(\mathrm{R\ loc}_1).(=) \sqcap \exists(\mathrm{loc}_2),(\mathrm{R\ loc}_2).(=) \sqcap \forall \mathrm{R.A}.$$

Concept A is satisfiable, as Figure 1 (left) shows. On the other hand, if this information is used while checking B, the algorithm returns that B is satisfiable, which is wrong, as can be seen in Figure 1 (right).

The problem is caused by the fact that the satisfiability of parts of a constraint network in general does not imply the satisfiability of the whole network. For this reason it is necessary to not only cache the label set, but also its concrete neighborhood.

Another problem arises from the fact that for a node $a$ and a parent node $b$ of $a$ the application of a rule on $a$ may trigger the application of a $\forall$-rule on $b$. For example consider a node $a$ with

$$\mathcal{L}(a) = \{\exists(\mathrm{loc}_1),(\mathrm{loc}_1).(=) \sqcap \exists(\mathrm{loc}_2),(\mathrm{loc}_2).(=)\}.$$

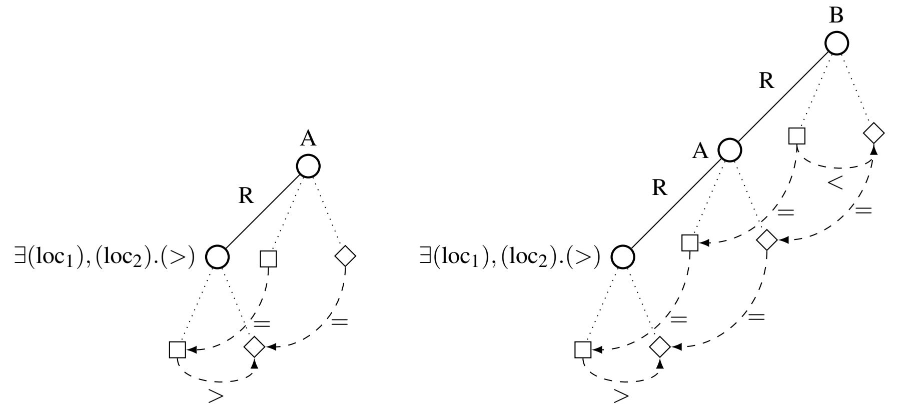
A
R
$\exists(\text{loc}_1),(\text{loc}_2).(>)$
=
=
>
B
R
A
R
$\exists(\text{loc}_1),(\text{loc}_2).(>)$
<
=
=
=
=
>

**Fig. 1.** Checking of concepts A (left) and B (right) of Example 1. The symbols □ and ◇ denote the concrete feature successors w.r.t. $\text{loc}_1$ and $\text{loc}_2$, resp.

and a parent node $b$ of $a$ connected by abstract feature $R$ with

$$\mathcal{L}(b) = \{\forall(\text{R loc}_1),(\text{R loc}_2).(\text{ntpp})\}.$$

Then, only if $a$ is completely expanded (the concrete successors are generated), the $\forall$-rule can be applied on $\mathcal{L}(b)$, i.e. the parent influences the concrete neighborhood of the child node. Our solution to this problem is to completely expand a node $a$ and its parent node $b$ before caching or loading the result for $a$.

Thus we restrict caching in our case in the two following ways:

– Only nodes that have an atomic and complete concrete neighborhood may be used for caching.
– Every node must be completely expanded before it is cached or used for loading of cached results.

This kind of caching is not as effective as caching for $\mathcal{ALC}$, because less nodes may be cached or loaded, and more rules must be applied before a node is considered for loading of a cached result.

## 3.2 Special techniques

In addition to these standard techniques, we use two optimizations tailored towards the usage of a constraint systems with general TBoxes. Both are suggested in [11] to be used in a particular fragment of $\mathcal{ALC}(\mathcal{C})$, but we discuss how one can use these in the general case.

**Patchwork property revisited.** In the original version of the algorithm in [11], the constructed constraint networks are atomic. As this requires to non-deterministically guess a relation for every disjunctive constraint, we will make use of a reformulation of the patchwork property (cp. [11]) to allow for disjunctive constraints in the constructed constraint network:

**Lemma 1.** *Let $\mathcal{C} = \langle D, \Gamma \rangle$ be a constraint system. Then the following statements are equivalent:*

*(i) $\mathcal{C}$ has the patchwork property.*
*(ii) For any finite networks $N$ and $N'$ such that $N|_{V_N \cap V_{N'}} = N'|_{V_N \cap V_{N'}}$ and $N|_{V_N \cap V_{N'}}$ is atomic and complete, if both $N$ and $N'$ are satisfiable, then so is the network $N \cup N'$.*

Notice the difference between statements (i) and (ii) of the lemma. Condition (ii) does not require the networks $N$ and $N'$ to be atomic and complete. This allows to refine networks to atomic networks only for concrete neighborhoods of potentially blocked nodes,i.e., the non-deterministic rules $\mathrm{R}\exists_c$ and $\mathrm{R}\forall_c$ can be replaced by deterministic versions that do not refine disjunctive constraints to atomic ones. The rationale here is that a constraint solver may handle disjunctive networks substantially faster than the non-optimized tableau algorithm. In particular the constraint solver can use search heuristics as well as tractability information of fragments of the constraint system. These techniques are not available on the part of the tableau reasoner, when constraint techniques are delegated to an external library. (This, by the way, is also the reason why the more general patchwork properties discussed in [9] can not be used in our setting).

Furthermore, since the network constructed during the tableau procedure needs to be checked for satisfiability after each step modifying the network, we can improve these checks by considering only that part of the network where a change occurs. The idea is to split the network into two parts that coincide on the concrete neighborhood of an abstract node (given that the concrete neighborhood is atomic and complete).

**Definition 1.** Let $(T, \mathcal{N})$ be a state in the tableau procedure, i.e., $T$ is some tree of abstract nodes and $\mathcal{N}$ the constructed constraint network. Let $a$ be an abstract node. The *succeeding network* of $a$ is defined by:

$\mathcal{N}_s(a) := \{(v\,r\,v') \in \mathcal{N}:$ there are concrete features $g, g'$ s.t. $v$ is $g$-successor of some abstract node $b$, $v'$ is $g'$-successor of some abstract node $b'$ where $b$ and $b'$ are successors of, or identical to, $a$ $\}$

The *preceding network* of $a$ is defined by:

$\mathcal{N}_p(a) := \{(v\,r\,v') \in \mathcal{N}:$ there are concrete features $g, g'$ s.t. $v$ is $g$-successor of some abstract node $b$, $v'$ is $g'$-successor of some abstract node $b'$, and $b$ and $b'$ are not successors of $a$ $\}$

**Lemma 2.** *In the situation of Definition 1, it holds:*

*(a) $\mathcal{N} = \mathcal{N}_s(a) \cup \mathcal{N}_p(a)$;*
*(b) $\mathcal{N}_s(a) \cap \mathcal{N}_p(a) = \mathcal{N}(a)$;*
*(c) $\mathcal{N}_s(a)|_{V_{\mathcal{N}_s(a)} \cap V_{\mathcal{N}_p(a)}} = \mathcal{N}_p(a)|_{V_{\mathcal{N}_s(a)} \cap V_{\mathcal{N}_p(a)}}$.*

**Example 3.** We consider the situation in Figure 2. The concrete neighborhood of node $b$ is atomic and complete (notice that each concrete node is related to itself via the identity relation). The preceding network is given by the relations $r_1$, $r_2$, $r_4$, $r_7$ and $r_8$, the

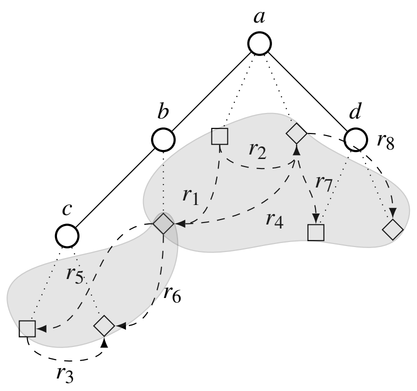

**Fig. 2.** Situation of Example 3. The symbols □ and ◇ represent different concrete features. The figure shows the preceding and succeeding networks of node $b$.

succeeding network by the relations $r_3$, $r_5$ and $r_6$. By Lemmas 1 and 2 the satisfiability of both the preceding and succeeding network ensures that the whole network is satisfiable.

By these lemmas it suffices at abstract nodes with a concrete neighborhood that is atomic and complete to divide the network into the preceding and the succeeding network (of the node). Of course, this separation method is reasonable for TBoxes with a single concrete feature, because then each concrete neighborhood is complete and atomic [11]. But also in the case of multiple concrete features, this separation method can be applied by first guessing an atomic completion of the concrete neighborhood of some abstract node $a$.

## 4 Implementation and evaluation

We implemented a reasoner based on the tableau algorithm and the optimization techniques presented in section 3. The reasoner is implemented in C++ and uses GQR [22] via libgqr for checking the satisfiability of spatial or temporal constraint networks. We decided to make the connection between GQR and our reasoner loose, i.e. we construct the network in our reasoner and translate it to a compatible data structure for the use in GQR. This design decision makes it possible to incorporate other constraint solvers without much effort.

To evaluate the proposed optimizations, we needed test problems. Unfortunately, there do not exist any benchmark problems or ontologies using the given approach based on $\mathcal{ALC}$ with constraints systems, which is maybe due to the fact that there does not exist a concrete language for writing such problems, either. For this reason we had to come up with synthetic test cases on our own.

To check the effect of the optimization techniques, we wanted to generate 'realistic' knowledge bases, and therefore we adapted the method proposed in [6]. The basic idea of this method is to analyze example knowledge bases in order to infer rules for the generation of further synthetic knowledge bases from those examples. In more detail, the method uses a fixed ratio of partial concept inclusions, a fixed number of role names, and a fixed scheme for concept definitions (in contrast to [6] we also use concept disjunction). Moreover, concepts are generated in layers to avoid terminological cycles (i.e., concept definitions used in layer $l$ mention concepts from the layers $0,\dots,l-1$ only). Further parameters are extracted from natural knowledge bases. For our experiments, we wrote some example knowledge bases from scratch and additionally adapted knowledge bases from SpacePortal on Ontohub [4]. In particular we considered the knowledge bases *ArchitecturalConstruction*[2] and *BuildingStructure*[3] that use RCC8-relations as roles and translated these into the $\mathcal{ALC}(\mathcal{C})$-framework. Definitions using constructors disallowed in $\mathcal{ALC}$ were rewritten or removed.

From these hand-crafted examples we derived the following rules:

- An axiom $\mathrm{A} \sqsubseteq \mathrm{B}$ or $\mathrm{A} \doteq \mathrm{B}$ contains concrete information (e.g. a conjunct $\forall(\mathrm{R\ loc}),(\mathrm{R\ loc}).(r_1 \vee \cdots \vee r_k)$ or $\exists(\mathrm{R\ loc}),(\mathrm{R\ loc}).(r_1 \vee \cdots \vee r_k)$) with a probability of 0.36.
- The average size of a constraint in a concrete concept is 3.31 in the case of Allen and 3.08 in the case of RCC8. So a relation $r$ is contained with a probability of $\frac{3.31}{13}$ in the case of Allen and $\frac{3.08}{8}$ in the case of RCC8.
- Only one concrete feature is used.
- Every declared concept is satisfiable, as in most real world knowledge bases this should be the case.

Notice that the knowledge bases generated in this way are in the fragment described in [11] as we only use one concrete feature. For each number 200, 400, ..., 2000 of axioms we generated 10 ontologies for both Allen and RCC8. For the evaluation we used a timeout of 300s per instance.

The following strategies for checking the satisfiability of all defined concepts (i.e. concepts on the left side of the axioms) were considered:

- *All Opts*: All considered optimizations were activated. This is the baseline, i.e., the other strategies differ from this strategy by deactivating single optimizations.
- *No Caching*: All optimizations are activated except for caching.
- *No Disjunctions*: Disjunctive constraints are resolved to atomic constraints by the tableau algorithm like in the original description of the algorithm in [11]. This is expected to be highly inefficient as there are no heuristics or other optimizations implemented for a careful selection of the relation, even though caching can be applied to every node as every concrete neighborhood is atomic and complete.
- *No Separation*: The patchwork property is not used to divide the network into smaller parts when the prerequisites of the patchwork property are satisfied. As we only allow one concrete feature, the deactivation of this optimization should have a huge impact on the runtime.

All tests were run on an Ubuntu 14.04 system with an Intel i3 U380 1.33GHz Processor and 4GB of RAM.

In Figure 3 the results of our tests are shown. As expected the runtimes of 'No Caching', 'No Disjunctions' and 'No Separation' are distinctly worse than the runtimes

[2] `http://ontohub.org/spaceportal/OntoSpace/ArchitecturalConstruction`
[3] `http://ontohub.org/spaceportal/OntoSpace/BuildingStructure`

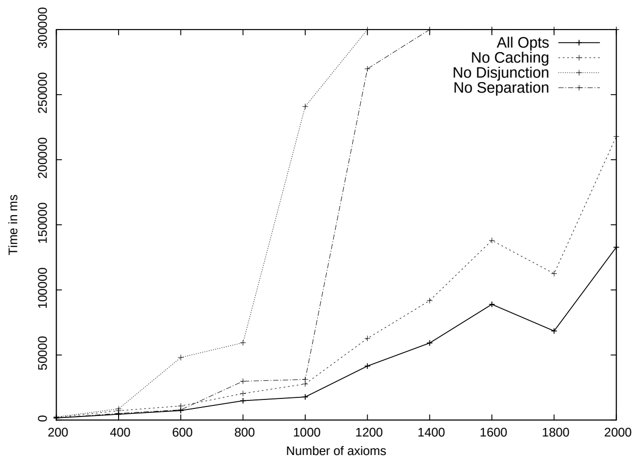

**Fig. 3.** Comparison of optimizations: 0.75-quantiles of the runtimes for generated knowledge bases using Allen interval constraints

of 'All Opts'. Notice that even the restricted form of caching used in our implementation shows to be beneficial. The most significant performance gains however are obtained by delegating disjunctive constraints to an external reasoner and by applying the generalized version of the patchwork property to split the network into smaller parts. Moreover, comparing the benefits of using all optimization techniques with respect to the used constraint system, our results do not show any significant difference between the constraint systems.

We also conducted some experiments on test instances with multiple concrete features. Our results indicate that instances without terminological cycles can be handled by our reasoner quite well, while the presence of terminological cycles may degrade its performance considerably.

## 5 Conclusion

We implemented and evaluated a reasoner for the description logic $\mathcal{ALC}(\mathcal{C})$ with general TBoxes and temporal and spatial constraint systems based on the algorithm proposed in [11]. To our knowledge, our implementation is the first reasoner supporting such constraint systems. We adapted optimizations from [8] to this case and applied optimizations discussed in [11] for a fragment of $\mathcal{ALC}(\mathcal{C})$ to the general case. Restrictions were necessary to use some of the optimizations, namely backjumping and caching. Our implementation is loosely connected to the used constraint solver, i.e. another constraint solver may be used without much effort. For example, in [12] a quite similar algorithm for checking concept satisfiability with general TBoxes and fuzzy concrete domains is proposed. It should be quite straightforward to adapt our implementation to this setting.

**Acknowledgements.** We thank the anonymous reviewers for helpful suggestions and comments. This work was supported by DFG (Transregional Collaborative Research Center *SFB/TR 8 Spatial Cognition*, project R4-[LogoSpace]).

## References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM 26(11), 832–843 (1983)
2. Baader, F., Hanschke, P.: A scheme for integrating concrete domains into concept languages. In: Mylopoulos, J., Reiter, R. (eds.) Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991. pp. 452–457. Morgan Kaufmann (1991)
3. Baader, F., Hollunder, B., Nebel, B., Profitlich, H.J., Franconi, E.: An empirical analysis of optimization techniques for terminological representation systems, or making KRIS get a move on. In: Nebel, B., Rich, C., Swartout, W.R. (eds.) Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, October 25-29, 1992. pp. 270–281. Morgan Kaufmann (1992)
4. Bateman, J., Kutz, O., Mossakowski, T., Sojic, A., Codescu, M.: Space for Space - Space-Portal: the 21st century home for spatial ontologies. In: Freksa, C., Nebel, B., Hegarty, M., Barkowsky, T. (eds.) Spatial Cognition 2014: Poster Presentations. pp. 13–16. No. 036-09/2014 in SFB/TR 8 Report (2014)
5. Grütter, R., Bauer-Messmer, B.: Towards spatial reasoning in the semantic web: A hybrid knowledge representation system architecture. In: Fabrikant, S.I., Wachowicz, M. (eds.) The European Information Society: Leading the Way with Geo-information, Proceedings of the 10th AGILE Conference, Aalborg, Denmark, 8-11 May 2007. pp. 349–364. Lecture Notes in Geoinformation and Cartography, Springer (2007)
6. Heinsohn, J., Kudenko, D., Nebel, B., Profitlich, H.: An empirical analysis of terminological representation systems. Artificial Intelligence 68(2), 367–397 (1994)
7. Horrocks, I.: Optimising Tableaux Decision Procedures for Description Logics. Ph.D. thesis, University of Manchester (1997)
8. Horrocks, I.: Implementation and optimization techniques. In: Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.) Description Logic Handbook. pp. 306–346. Cambridge University Press (2003)
9. Huang, J.: Compactness and its implications for qualitative spatial and temporal reasoning. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012. AAAI Press (2012)
10. Lutz, C.: Description logics with concrete domains – a survey. In: Balbiani, P., Suzuki, N., Wolter, F., Zakharyaschev, M. (eds.) Advances in Modal Logic 4, papers from the fourth conference on "Advances in Modal logic," held in Toulouse (France) in October 2002. pp. 265–296. King's College Publications (2002)
11. Lutz, C., Miličić, M.: A tableau algorithm for description logics with concrete domains and general TBoxes. Journal of Automated Reasoning 38(1-3), 227–259 (2007)
12. Merz, D., Peñaloza, R., Turhan, A.: Reasoning in $\mathcal{ALC}$ with fuzzy concrete domains. In: Lutz, C., Thielscher, M. (eds.) KI 2014: Advances in Artificial Intelligence - 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Lecture Notes in Computer Science, vol. 8736, pp. 171–182. Springer (2014)
13. Randell, D.A., Cui, Z., Cohn, A.G.: A spatial logic based on regions and connection. In: Nebel, B., Rich, C., Swartout, W.R. (eds.) Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, October 25-29, 1992. pp. 165–176. Morgan Kaufmann (1992)
14. Renz, J., Nebel, B.: Qualitative spatial reasoning using constraint calculi. In: Aiello, M., Pratt-Hartmann, I., van Benthem, J. (eds.) Handbook of Spatial Logics, pp. 161–215. Springer (2007)

15. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artificial Intelligence 48(1), 1–26 (1991)
16. Shearer, R., Motik, B., Horrocks, I.: HermiT: A highly-efficient OWL reasoner. In: Dolbear, C., Ruttenberg, A., Sattler, U. (eds.) Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions, collocated with the 7th International Semantic Web Conference (ISWC-2008), Karlsruhe, Germany, October 26-27, 2008. CEUR Workshop Proceedings, vol. 432. CEUR-WS.org (2008)
17. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web 5(2), 51–53 (2007)
18. Stocker, M., Sirin, E.: PelletSpatial: A hybrid RCC-8 and RDF/OWL reasoning and query engine. In: Hoekstra, R., Patel-Schneider, P.F. (eds.) Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009. CEUR Workshop Proceedings, vol. 529. CEUR-WS.org (2009)
19. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Furbach, U., Shankar, N. (eds.) Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006. Lecture Notes in Computer Science, vol. 4130, pp. 292–297. Springer (2006)
20. Turhan, A., Haarslev, V.: Adapting optimization techniques to description logics with concrete domains. In: Baader, F., Sattler, U. (eds.) Proceedings of the 2000 International Workshop on Description Logics (DL2000), Aachen, Germany, August 17-19, 2000. CEUR Workshop Proceedings, vol. 33, pp. 247–256. CEUR-WS.org (2000)
21. Wessel, M.: Obstacles on the way to qualitative spatial reasoning with description logics: Some undecidability results. In: Goble, C.A., McGuinness, D.L., Möller, R., Patel-Schneider, P.F. (eds.) Working Notes of the 2001 International Description Logics Workshop (DL-2001), Stanford, CA, USA, August 1-3, 2001. CEUR Workshop Proceedings, vol. 49. CEUR-WS.org (2001)
22. Westphal, M., Wölfl, S., Gantner, Z.: GQR: A fast solver for binary qualitative constraint networks. In: Benchmarking of Qualitative Spatial and Temporal Reasoning Systems, Papers from the 2009 AAAI Spring Symposium, Technical Report SS-09-02, Stanford, California, USA, March 23-25, 2009. pp. 51–52. AAAI (2009)