ulm university universität
uulm

Extended Caching, Backjumping and Merging for Expressive Description Logics

Andreas Steigmiller, Thorsten Liebig, Birte Glimm

Ulmer Informatik-Berichte

Nr. 2012-01
Mai 2012

Extended Caching, Backjumping and Merging for Expressive Description Logics

Andreas Steigmiller[1], Thorsten Liebig[2], and Birte Glimm[1]

[1] Ulm University, Ulm, Germany, <first name>.<last name>@uni-ulm.de
[2] derivo GmbH, Ulm, Germany, liebig@derivo.de

Abstract. With this contribution we push the boundary of some known optimisations such as caching to the very expressive Description Logic $\mathcal{SROIQ}$. The developed method is based on a sophisticated dependency management and a precise unsatisfiability caching technique, which further enables better informed tableau backtracking and more efficient pruning. Additionally, we optimise the handling of cardinality restrictions, by introducing a strategy called pool-based merging.
We empirically evaluate the proposed optimisations within the novel reasoning system Konclude and show that the proposed optimisations indeed result in significant performance improvements.

1 Motivation

Tableau algorithms are dominantly used in sound and complete reasoning systems, which are able to deal with ontologies specified in the OWL 2 DL ontology language [18]. Such algorithms are usually specified in terms of Description Logics (DLs) [1], which provide the formal basis for OWL, e.g., OWL 2 is based on the DL $\mathcal{SROIQ}$ [12].

To our knowledge, all competitive systems for reasoning with $\mathcal{SROIQ}$ knowledge bases such as FaCT++ [20], HermiT,[3] jFact,[4] or Pellet [19] use a variant of the tableau method – a refutation-based calculus that systematically tries to construct an abstraction of a model for a given query by exhaustive application of so called tableau rules.

Due to the wide range of modelling constructs supported by expressive DLs, the typically used tableau algorithms have a very high worst-case complexity. Developing optimisations to nevertheless allow for highly efficient implementations is, therefore, a long-standing research area in DLs (see, e.g., [14,21]). A very effective and widely implemented optimisation technique is "caching", where one caches, for a set of concepts, whether they are known to be, or can safely be assumed to be, satisfiable or unsatisfiable [4]. If the set of concepts appears again in a model abstraction, then a cache-lookup allows for skipping further applications of tableau rules. Caching even allows for implementing worst-case optimal decision procedures for $\mathcal{ALC}$ [6].

Unfortunately, with increasing expressivity some of the widely used optimisations become unsound. For instance, naively caching the satisfiability status of interim results

[3] `http://www.hermit-reasoner.com`
[4] `http://jfact.sourceforge.net/`

easily causes unsoundness in the presence of inverse roles due to their possible interactions with universal restrictions [1, Chapter 9]. On the other hand, for features such as cardinality restrictions there are nearly no optimisations yet. An attempt to use algebraic methods [10,5], i.e., by combining a tableau calculus with a procedure to solve systems of linear (in)equations, performs well, but requires significant changes to the calculus and has not (yet) been extended to very expressive DLs such as $\mathcal{SROIQ}$.

Our contribution in this paper is two-fold. We push the boundary of known optimisations, most notably caching, to the expressive DL $\mathcal{SROIQ}$. The developed method is based on a sophisticated dependency management and a precise unsatisfiability caching technique, which further enables better informed tableau backtracking and more efficient pruning (Section 3). In addition we optimise the handling of cardinality restrictions, by introducing a strategy called *pool-based merging* (Section 4). Our techniques are grounded in the widely implemented tableau calculus for $\mathcal{SROIQ}$ [12], which makes it easy to transfer our results into existing tableau implementations. The presented optimisations are implemented within a novel reasoning system, called Konclude [17]. Our empirical evaluation shows that the proposed optimisations result in significant performance improvements (Section 5).

2 Preliminaries

Model construction calculi, such as tableau, decide the consistency of a knowledge base $\mathcal{K}$ by trying to construct an abstraction of a model for $\mathcal{K}$, a so-called "completion graph". A completion graph G is a tuple $(V, E, \mathcal{L}, \dot{\neq})$, where each node $x \in V$ represents one or more individuals, and is labelled with a set of concepts, $\mathcal{L}(x)$, which the individuals represented by x are instances of; each edge $\langle x, y \rangle$ represents one or more pairs of individuals, and is labelled with a set of roles, $\mathcal{L}(\langle x, y \rangle)$, which the pairs of individuals represented by $\langle x, y \rangle$ are instances of. The relation $\dot{\neq}$ records inequalities, which must hold between nodes, e.g., due to at-least cardinality restrictions.

The algorithm works by initialising the graph with one node for each Abox individual/nominal in the input KB, and using a set of expansion rules to syntactically decompose concepts in node labels. Each such rule application can add new concepts to node labels and/or new nodes and edges to the completion graph, thereby explicating the structure of a model. The rules are repeatedly applied until either the graph is fully expanded (no more rules are applicable), in which case the graph can be used to construct a model that is a *witness* to the consistency of $\mathcal{K}$, or an obvious contradiction (called a *clash*) is discovered (e.g., both C and $\neg C$ in a node label), proving that the completion graph does not correspond to a model. The input knowledge base $\mathcal{K}$ is *consistent* if the rules (some of which are non-deterministic) can be applied such that they build a fully expanded, clash free completion graph. A cycle detection technique called *blocking* ensures the termination of the algorithm.

2.1 Dependency Tracking

Dependency tracking keeps track of all dependencies that cause the existence of concepts in node labels, roles in edge labels as well as accompanying constrains such as

inequalities that must hold between nodes. Dependencies are associated with so-called *facts*, defined as follows:

Definition 1 (Fact) *We say that G contains a* concept fact $C(x)$ *if* $x \in V$ *and* $C \in \mathcal{L}(x)$*, G contains a* role fact $r(x, y)$ *if* $\langle x, y\rangle \in E$ *and* $r \in \mathcal{L}(\langle x, y\rangle)$*, and G contains an* inequality fact $x \mathrel{\dot{\neq}} y$ *if* $x, y \in V$ *and* $(x, y) \in \mathrel{\dot{\neq}}$*. We denote the set of all (concept, role, or inequality) facts in G as* Facts_G*.*

Dependencies now relate facts in a completion graph to the facts that caused their existence. Additionally, we annotate these relations with a running index, called dependency number, and a branching tag to track non-deterministic expansions:

Definition 2 (Dependency) *Let d be a pair in* $\mathsf{Facts}_G \times \mathsf{Facts}_G$*. A* dependency *is of the form* $d^{n,b}$ *with* $n \in \mathbf{N}_0$ *a* dependency number *and* $b \in \mathbf{N}_0$ *a* branching tag.

We inductively define the dependencies for *G, written* Dep_G*. If G is an initial completion graph, then* $\mathsf{Dep}_G = \emptyset$*. Let R be a tableau rule applicable to a completion graph G with* $\{c_0, \ldots, c_k\}$ *a minimal set of facts in G that satisfy the preconditions of R. If* $\mathsf{Dep}_G = \emptyset$*, then* $n_m = b_m = 0$*, otherwise, let* $n_m = max\{n \mid d^{n,b} \in \mathsf{Dep}_G\}$ *and* $b_m = max\{b \mid d^{n,b} \in \mathsf{Dep}_G\}$*. If R is non-deterministic, then* $b_R = 1 + b_m$*, otherwise* $b_R = 0$*. Let G′ be the completion graph obtained from G by applying R and let* $c'_0, \ldots, c'_\ell$ *be the newly added facts in G′, then*

$$\mathsf{Dep}_{G'} = \mathsf{Dep}_G \cup \{(c'_j, c_i)^{n,b} \mid 0 \le i \le k, 0 \le j \le \ell, n = n_m + 1 + (j * k) + i, \\ b = max\{\{b_R\} \cup \{b' \mid (c_i, c)^{n',b'} \in \mathsf{Dep}_G\}\}\}.$$

The branching tag indicates which facts were added non-deterministically:

Definition 3 (Non-deterministic Dependency) *For* $d^{n,b} \in \mathsf{Dep}_G$ *with* $d = (c_1, c_2)$*, let* $D_d = \{(c_2, c_3)^{n',b'} \mid (c_2, c_3)^{n',b'} \in \mathsf{Dep}_G\}$*. The dependency* $d^{n,b}$ *is a* non-deterministic dependency in *G if* $b > 0$ *and either* $D_d = \emptyset$ *or* $max\{b' \mid (c, c')^{n',b'} \in D_d\} < b$*.*

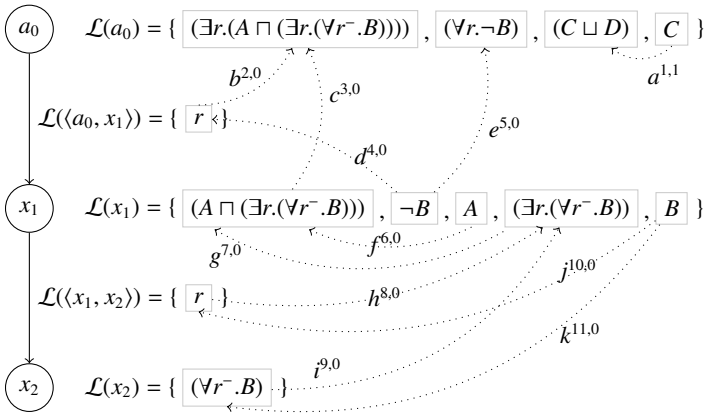

Fig. 1. Tracked dependencies for all facts in the generated completion graph

Figure 1 illustrates a completion graph obtained in the course of testing the consistency of a knowledge base with three concept assertions:

$$a_0 : (\exists r.(A \sqcap (\exists r.(\forall r^{-}.B)))) \qquad a_0 : (\forall r.\neg B) \qquad a_0 : (C \sqcup D).$$

Thus, the completion graph is initialised with the node a_0, which has the three concepts in its label. Initially, the set of dependencies is empty. For the concepts and roles added by the application of tableau rules, the dependencies are shown with dotted lines, labelled with the dependency. The dependency number increases with every new dependency. The branching tag is only non-zero for the non-deterministic addition of C to the label of a_0 in order to satisfy the disjunction $(C \sqcup D)$. Note the presence of a clash due to B and $\neg B$ in the label of x_1.

3 Extended Caching and Backtracking

In the following we introduce improvements to caching and backjumping by presenting a more informed dependency directed backtracking strategy that also allows for extracting precise unsatisfiability cache entries.

3.1 Dependency Directed Backtracking

Dependency directed backtracking is an optimisation that can effectively prune irrelevant alternatives of non-deterministic branching decisions. If branching points are not involved in clashes, it will not be necessary to compute any more alternatives of these branching points, because the other alternatives cannot eliminate the cause of the clash. To identify involved non-deterministic branching points, all facts in a completion graph are labelled with information about the branching points they depend on. Thus, the united information of all clashed facts can be used to identify involved branching points. A typical realisation of dependency directed backtracking is backjumping [1,21], where the dependent branching points are collected in the dependency sets for all facts.

3.2 Unsatisfiability Caching

Another widely used technique to increase the performance of a tableau implementation is caching, which comes in two flavours: satisfiability and unsatisfiability caching. For the former, one caches sets of concepts, e.g., from node labels, that are known to be satisfiable. In contrast, for an unsatisfiability cache, we cache sets of concepts that are unsatisfiable. For such a cached set, any *superset* is also unsatisfiable. Thus, one is interested in caching a minimal, unsatisfiable set of concepts. Although the caching of satisfiable and unsatisfiable sets of concepts is often considered together, we focus here on the unsatisfiability caching problem since the two problems are quite different in nature and already the required data structure for an efficient cache retrieval can differ significantly.

Definition 4 (Unsatisfiability Cache) *Let $\mathcal{K}$ be a knowledge base, $\mathsf{Con}_{\mathcal{K}}$ the set of (sub-)concepts that occur in $\mathcal{K}$. An* unsatisfiability cache $\mathsf{UC}_{\mathcal{K}}$ *for $\mathcal{K}$ is a subset of*

$2^{Con_\mathcal{K}}$ *such that each* cache entry $S \in \mathsf{UC}_\mathcal{K}$ *is an unsatisfiable set of concepts. An* unsatisfiability retrieval *for* $\mathsf{UC}_\mathcal{K}$ *and a completion graph* G *for* $\mathcal{K}$ *takes a set of concepts* $S \subseteq Con_\mathcal{K}$ *from a node label of* G *as input. If* $\mathsf{UC}_\mathcal{K}$ *contains a set* $S_\bot \subseteq S$*, then* $S_\bot$ *is returned; otherwise, the empty set is returned.*

Deciding when we can safely create a cache entry rapidly becomes difficult with increasing expressivity of the used DL. Already with blocking on tableau-based systems for the DL $\mathcal{ALC}$ care has to be taken to not generate invalid cache entries [8]. There are some approaches for caching with inverse roles [2,3,6], where possible propagations over inverse roles from descendant nodes are taken into account. The difficulty increases further in the presence of nominals and, to the best of our knowledge, the problem of caching with inverses and nominals has not yet been addressed in the literature. In this setting, it is difficult to determine, for a node x with a clash in its label, which nodes (apart from x) are also labelled with unsatisfiable sets of concepts. Without nominals and inverse roles, we can determine the ancestor y of x with the last non-deterministic expansion and consider the labels of all nodes from x up to y as unsatisfiable. With inverse roles, a non-deterministic rule application on a descendant node of x can be involved in the creation of the clash, whereby the node labels that can be cached as unsatisfiable become limited.

In order to demonstrate the difficulties with inverse roles, let us assume that the example in Figure 1 is extended such that $((\forall r^-.B) \sqcup E) \in \mathcal{L}(x_2)$ and that $(\forall r^-.B) \in \mathcal{L}(x_2)$ results from the non-deterministic expansion of the disjunction. For the resulting clash in $\mathcal{L}(x_1)$, it is not longer sufficient to consider only non-deterministic expansions on ancestor nodes. The label of x_2 cannot be cached because some facts $(\neg B)$ involved in the clash are located on different nodes (x_1). Furthermore, if trying the disjunct E also leads to a clash, the disjunction $((\forall r^-.B) \sqcup E)$ in $\mathcal{L}(x_2)$ is unsatisfiable in the context of *this* completion graph. Nevertheless, a cache entry cannot be generated because (at least) the first disjunct involves facts of an ancestor node. In order to also handle inverse roles, it would, therefore, be necessary to remember all nodes or at least the minimum node depth involved in the clashes of all alternatives. In the presence of nominals, it further becomes necessary to precisely manage the exact causes of clashes, e.g., via tracking the dependencies as presented in Section 2.1. If such a technique is missing, often the only option is to deactivate caching completely [19,21].

Since node labels can have many concepts that are not involved in any clashes, the precise extraction of a small set of concepts that are in this combination unsatisfiable would yield better entries for the unsatisfiability cache. With an appropriate subset retrieval potentially more similar also unsatisfiable node labels can be found within the cache. We call this technique *precise caching*. Although techniques to realise efficient subset retrieval exist [11], unsatisfiability caches based on this idea are only implemented in very few DL reasoners [9]. Furthermore, the often used backjumping only allows the identification of all branching points involved in a clash, but there is no information about how the clash is exactly caused. As a result, only complete node labels can be saved in the unsatisfiability cache. We refer to this often used form of caching combined with only an equality cache retrieval as *label caching*.

For precise caching, the selection of an as small as possible but still unsatisfiable subset of a label as cache entry should be adjusted to the cache retrieval strategy, i.e.,

the strategy of when the cache is queried in the tableau algorithm. Going back to the example in Figure 1, for the node x_1 the set $\{\neg B, (\exists r.(\forall r^{-}.B))\}$ could be inserted into the cache as well as $\{\neg B, (A \sqcap (\exists r.(\forall r^{-}.B)))\}$. The number of cache entries should, however, be kept small, because the performance of the retrieval decreases with an increasing number of entries. Thus, the insertion of concepts for which the rule application is cheap (e.g., concept conjunction) should be avoided. Concepts that require the application of non-deterministic or generating rules are more suitable, because the extra effort of querying the unsatisfiability cache before the rule application can be worth the effort. Optimising cache retrievals for incremental changes further helps to efficiently handle multiple retrievals for the same node with identical or slightly extended concept labels.

The creation of new unsatisfiability cache entries based on dependency tracking can be done during backtracing, which is also coupled with the dependency directed backtracking as described next. Basically all facts involved in a clash are backtraced to collect the facts that cause the clash within one node, whereby then an unsatisfiability cache entry can be created.

3.3 Dependency Backtracing

The dependency tracking defined in Section 2.1 completely retains all necessary information to exactly trace back the cause of the clash. Thus, this *backtracing* is qualified to identify all involved non-deterministic branching points for the dependency directed backtracking and also to identify small unsatisfiable sets of concepts that can be used to create new unsatisfiability cache entries.

Algorithm 1 performs the backtracing of facts and their tracked dependencies in the presence of inverse roles and nominals. If all facts and their dependencies are collected on the same node while backtracing, an unsatisfiability cache entry with these facts can be generated, assuming all facts are concept facts. As long as no nominal or Abox individual occurs in the backtracing, the unsatisfiability cache entries can also be generated while all concept facts have the same node depth. Thus, an important task of the backtracing algorithm is to hold as many facts as possible within the same node depth to allow for the generation of many cache entries. To realise the backtracing, we introduce the following data structure:

Definition 5 (Fact Dependency Node Tuple) *A* fact dependency node tuple for G *is a triple* $\langle c, d^{n,b}, x\rangle$ *with* $c \in \mathsf{Facts}_G, d^{n,b} \in \mathsf{Dep}_G$ *and* $x \in V$. *Abbreviatory we also write* $\langle C, d^{n,b}, x\rangle$ *if* c *is the concept fact* $C(x)$.

If a clash is discovered in the completion graph, a set of fact dependency node tuples is generated for the backtracing. Each tuple consists of a fact involved in the clash, an associated dependency and the node where the clash occurred. The algorithm gets this set T of tuples as input and incrementally traces the facts back from the node with the clash to nodes with depth 0 (Abox individuals or root nodes).

In each loop round (line 3) some tuples of T are exchanged with tuples, whose facts are the cause of the exchanged one. To identify which tuple has to be traced back first, the current minimum node depth (line 4) and the maximum branching tag (line 5) are extracted from the tuples of T. All tuples, whose facts are located on a deeper node and

Algorithm 1 Backtracing Algorithm

Require: A set of fact dependency node tuples T obtained from clashes

procedure DEPENDENCYBACKTRACING(T)
  $pendingUnsatCaching \leftarrow false$
  loop
    $min_D \leftarrow$MINIMUMNODEDEPTH(T)
    $max_B \leftarrow$MAXIMUMBRANCHINGTAG(T)
    $A \leftarrow \{t \in T \mid$ NODEDEPTH(t)$> min_D \wedge$ HASDETERMINISTICDEPENDENCY(t)$\}$
    $C \leftarrow \emptyset$
    if $A \neq \emptyset$ **then**
      $pendingUnsatCaching \leftarrow true$
      for all $t \in A$ **do**
        $T \leftarrow (T \setminus t) \cup$ GETCAUSETUPLESBYDEPENDENCY(t)
      end for
    else
      $B \leftarrow \{t \in T \mid$ NODEDEPTH(t)$> min_D \wedge$ BRANCHINGTAG(t)$= max_B\}$
      if $B = \emptyset$ **then**
        if $pendingUnsatCaching = true$ **then**
          $pendingUnsatCaching \leftarrow$TRYCREATEUNSATCACHEENTRY(T)
        end if
        if HASNODEPENDENCY(t) **for all** $t \in T$ **then**
          $pendingUnsatCaching \leftarrow$TRYCREATEUNSATCACHEENTRY(T)
          return
        end if
        $C \leftarrow \{t \in T \mid$ BRANCHINGTAG(t)$= max_B\}$
      end if
      $t \leftarrow$ANYELEMENT($B \cup C$)
      if HASDETERMINISTICDEPENDENCY(t) **then**
        $T \leftarrow (T \setminus t) \cup$ GETCAUSETUPLESBYDEPENDENCY(t)
      else
        $b \leftarrow$GETNONDETERMINISTICBRANCHINGPOINT(t)
        if ALLALTERNATIVESOFNONDETBRANCHINGPOINTPROCESSED(b) **then**
          $T \leftarrow T \cup$ LOADTUPLESFROMNONDETBRANCHINGPOINT(b)
          $T \leftarrow (T \setminus t) \cup$ GETCAUSETUPLESBYDEPENDENCY(t)
          $T \leftarrow$FORCETUPLESBEFOREBRANCHINGPOINT(T, b)
          $pendingUnsatCaching \leftarrow$TRYCREATEUNSATCACHEENTRY(T)
        else
          $T \leftarrow$FORCETUPLESBEFOREBRANCHINGPOINT(T, b)
          SAVETUPLESTONONDETBRANCHINGPOINT(T, b)
          JUMPBACKTO(max_B)
          return
        end if
      end if
    end if
  end loop
end procedure

whose dependencies are deterministic, are collected in the set A. Such tuples will be directly traced back until their facts reach the current minimum node depth (line 10-12). If there are no more tuples on deeper nodes with deterministic dependencies, i.e., $A = \emptyset$, the remaining tuples from deeper nodes with non-deterministic dependencies and the current branching tag are copied into B (line 14) in the next round. If B is not empty, one of these tuples (line 25) and the corresponding non-deterministic branching point (line 29) are processed. The backtracing is only continued, if all alternatives of the branching point are computed as unsatisfiable. In this case, all tuples, saved from the backtracing of other unsatisfiable alternatives, are added to T (line 31). Moreover, for c the fact in t, t can be replaced with tuples for the fact on which c non-deterministically depends (line 32).

For a possible unsatisfiability cache entry all remaining tuples, which also depend on the non-deterministic branching point, have to be traced back until there are no tuples with facts of some alternatives of this branching point left (line 33). An unsatisfiability cache entry is only generated (line 34), if all facts in T are concept facts for the same node or on the same node depth.

Unprocessed alternatives of a non-deterministic branching point have to be computed before the backtracing can be continued. It is, therefore, ensured that tuples do not consist of facts and dependencies from this alternative, which also allows for releasing memory (line 36). The tuples are saved to the branching point (line 37) and the algorithm jumps back to an unprocessed alternative (line 38).

If B is also empty, but there are still dependencies to previous facts, some tuples based on the current branching tag have to remain on the current minimum node depth. These tuples are collected in the set C (line 23) and are processed separately one per loop round, similar to the tuples of B, because the minimum node depth or maximum branching tag may change. The tuples of C can have deterministic dependencies, which are processed like the tuples of A (line 27). If all tuples have no more dependencies to previous facts, the algorithm terminates (line 21).

Besides the creation of unsatisfiability cache entries after non-deterministic dependencies (line 34), cache entries may also be generated when switching from a deeper node to the current minimum node depth in the backtracing (line 9 and 17) or when the backtracing finishes (line 20). The function that tries to create new unsatisfiability cache entries (line 17, 20, and 34) returns a Boolean flag that indicates whether the attempt has failed, so that the attempt can be repeated later.

For an example, we consider the clash $\{\neg B, B\}$ in the completion graph of Figure 1. The initial set of tuples for the backtracing is T_1 (see Figure 2). Thus, the minimum node depth for T_1 is 1 and the maximum branching tag is 0. Because there are no tuples on a deeper node, the sets A and B are empty for T_1. Since all clashed facts are generated deterministically, the dependencies of the tuples have the current maximum branching tag 0 and are all collected into the set C. The backtracing continues with one tuple t from C, say $t = \langle B, k^{11,0}, x_1 \rangle$. The dependency k of t relates to the fact $(\forall r^-.B)(x_2)$, which is a part of the cause and replaces the backtraced tuple t in T_1. The resulting set T_2 is used in the next loop round. The minimum node depth and the maximum branching tag remain unchanged, but the new tuple has a deeper node depth and is traced back with a higher priority to enable unsatisfiability caching again. Thus, $\langle (\forall r^-.B), i^{9,0}, x_2 \rangle$ is added to the

$$
\begin{aligned}
T_1 &= \{\langle \neg B, d^{4,0}, x_1\rangle, \langle \neg B, e^{5,0}, x_1\rangle, \langle B, j^{10,0}, x_1\rangle, \langle B, k^{11,0}, x_1\rangle\} \\
&\downarrow \\
T_2 &= \{\langle \neg B, d^{4,0}, x_1\rangle, \langle \neg B, e^{5,0}, x_1\rangle, \langle B, j^{10,0}, x_1\rangle, \langle (\forall r^{-}.B), i^{9,0}, x_2\rangle\} \\
&\downarrow \\
T_3 &= \{\langle \neg B, d^{4,0}, x_1\rangle, \langle \neg B, e^{5,0}, x_1\rangle, \langle B, j^{10,0}, x_1\rangle, \langle (\exists r.(\forall r^{-}.B)), g^{7,0}, x_1\rangle\} \\
&\downarrow \\
T_4 &= \{\langle \neg B, d^{4,0}, x_1\rangle, \langle \neg B, e^{5,0}, x_1\rangle, \langle r(x_1, x_2), h^{8,0}, x_1\rangle, \langle (\exists r.(\forall r^{-}.B)), g^{7,0}, x_1\rangle\} \\
&\downarrow \\
T_5 &= \{\langle \neg B, d^{4,0}, x_1\rangle, \langle \neg B, e^{5,0}, x_1\rangle, \langle (\exists r.(\forall r^{-}.B)), g^{7,0}, x_1\rangle\} \\
&\downarrow \\
T_6 &= \{\langle \neg B, d^{4,0}, x_1\rangle, \langle (\forall r.\neg B), -, a_0\rangle, \langle (\exists r.(\forall r^{-}.B)), g^{7,0}, x_1\rangle\} \\
&\downarrow \\
T_7 &= \{\langle r(a_0, x_1), b^{2,0}, x_1\rangle, \langle (\forall r.\neg B), -, a_0\rangle, \langle (A \sqcap (\exists r.(\forall r^{-}.B))), c^{3,0}, x_1\rangle\} \\
&\downarrow \\
T_8 &= \{\langle (\exists r.(A \sqcap (\exists r.(\forall r^{-}.B)))) -, a_0\rangle, \langle (\forall r.\neg B), -, a_0\rangle\}
\end{aligned}
$$

Fig. 2. Backtracing sequence of tuples as triggered by the clash of Figure 1

set A and then replaced by its cause, leading to T_3. Additionally, a pending creation of an unsatisfiability cache entry is noted, which is attempted in the third loop round since A and B are empty. The creation of a cache entry is, however, not yet sensible and deferred since T_3 still contains an atomic clash. Let $t = \langle B, j^{10,0}, x_1\rangle \in C$ be the tuple from T_3 that is traced back next. In the fourth round, the creation of a cache entry is attempted again, but fails because not all facts are concepts facts. The backtracing of $\langle r(x_1, x_2), h^{8,0}, x_1\rangle$ then leads to T_5. In the following round an unsatisfiability cache entry is successfully created for the set $\{\neg B, (\exists r.(\forall r^{-}.B))\}$. Assuming that now the tuple $\langle \neg B, e^{5,0}, x_1\rangle$ is traced back, we obtain T_6, which includes the node a_0. Thus, the minimum node depth changes from 1 to 0. Two more rounds are required until T_8 is reached. Since all remaining facts in T_8 are concept assertions, no further backtracing is possible and an additional cache entry is generated for the set $\{(\exists r.(A \sqcap (\exists r.(\forall r^{-}.B)))), (\forall r.\neg B)\}$.

If a tuple with a dependency to node a_0 had been traced back first, it would have been possible that the first unsatisfiability cache entry for the set $\{\neg B, (\exists r.(\forall r^{-}.B))\}$ was not generated. In general, it is not guaranteed that an unsatisfiability cache entry is generated for the node where the clash is discovered if there is no non-deterministic rule application and if the node is not a root node or an Abox individual. Furthermore, if there are facts that are not concept facts, these can be backtraced with higher priority, analogous to the elements of the set A, to make unsatisfiability cache entries possible again. To reduce the repeated backtracing of identical tuples in different rounds, an additional set can be used to store processed tuples for the alternative for which the backtracing is performed.

The backtracing can also be performed over nominal and Abox individual nodes. However, since Abox and absorbed nominal assertions such as $\{a\} \sqsubseteq C$ have no previous dependencies, this can lead to a distributed backtracing stuck on different nodes. In this case, no unsatisfiability cache entries are possible.

A less precise caching can lead to an adverse interaction with dependency directed backtracking. Consider the example of Figure 3, where the satisfiability of the combination of the concepts $(\exists r.(\exists s.(A \sqcap B)))$, $((C_1 \sqcap \forall r.C) \sqcup (C_2 \sqcap \forall r.C))$, and $((D_1 \sqcap \forall r.(\forall s.\neg A)) \sqcup (D_2 \sqcap \forall r.(\forall s.\neg A)))$ is tested. Note that, in order to keep the figure readable, we no longer show complete dependencies, but only the branching points for non-deterministic deci-

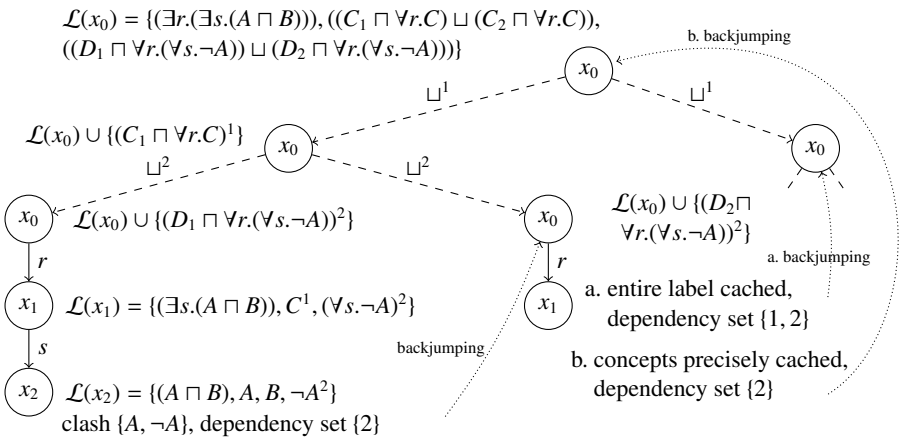

Fig. 3. More pruned alternatives due to dependency directed backtracking and precise caching (b.) in contrast to label caching (a.)

sions. First, the two disjunctions are processed. Assuming that the alternative with the disjuncts $(C_1 \sqcap \forall r.C)$ and $(D_1 \sqcap \forall r.(\forall s.\neg A))$ is considered first (shown on the left-hand side of Figure 3), an r-successor x_1 with label $\{(\exists s.(A \sqcap B)), C^1, (\forall s.\neg A)^2\}$ is generated. The branching points indicate which concepts depend on which non-deterministic decision. For example, C is in $\mathcal{L}(x_1)$ due to the disjunct $(C_1 \sqcap \forall r.C)$ of the first non-deterministic branching decision (illustrated in Figure 3 with the superscript 1). In the further generated s-successor x_2 a clash is discovered. For the only involved non-deterministic branching point 2, we have to compute the second alternative. Thus, an identical r-successor x_1 is generated again for which we can discover the unsatisfiability with a cache retrieval. If the entire label of x_1 was inserted to the cache, the dependent branching points of all concepts in the newly generated node x_1 would have to be considered for further dependency directed backtracking. Thus, the second alternative of the first branching decision also has to be evaluated (c.f. Figure 3, a.). In contrast, if the caching was more precise and only the combination of the concepts $(\exists s.(A \sqcap B))$ and $(\forall s.\neg A)$ was inserted into the unsatisfiability cache, the cache retrieval for the label of node x_1 would return the inserted subset. Thus, only the dependencies associated to the concepts of the subset could be used for further backjumping, whereby it would not be necessary to evaluate the remaining alternatives (c.f. Figure 3, b.).

4 Optimised Merging

At-most cardinality restrictions require the non-deterministic merging of role neighbours until the cardinality restriction is satisfied. Only for cardinalities of 1, merging is deterministic. The usual merging approach [12], which can still be found in several available reasoner implementations, employs a $\leq$-rule that shrinks the number of role neighbours by one with each rule application. Each such merging step gathers pairs of

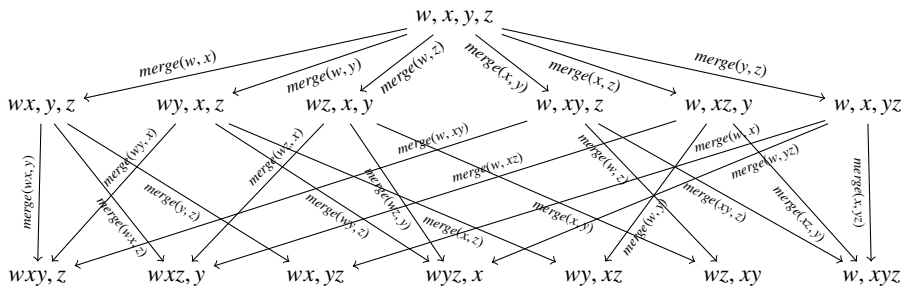

Fig. 4. Non-deterministic alternatives for pair-based merging

potentially mergeable neighbouring nodes. For each merging pair a branch is generated in which the merging of the pair is executed. Without optimisations, this approach leads to an inefficient implementation since for merging problems that require more than one merging step, several identical merging combinations have to be evaluated multiple times. Throughout this section, we consider the following example: a node in the completion graph has four r-neighbours w, x, y and z, which have to be merged into two nodes. The naive approach described above leads to eighteen non-deterministic alternatives (c.f. Figure 4): in the first of two necessary merging steps there are $\sum_{i=1}^{n-1} i$, i.e., six possible merging pairs. A second merging step is required to reduce the remaining three nodes to two. If the merging rule is applied again without any restrictions, each second merging step generates three more non-deterministic alternatives. However, only seven of these eighteen alternatives overall are really different. For example, the combination wxy, z, where the nodes w, x and y have been merged, can be generated by $merge(merge(w, x), y)$, $merge(merge(w, y), x)$ and $merge(merge(x, y), w)$.

The amount of redundant alternatives depends also strongly on the test cases and on the strategy of rule application. Applying the $\leq$-rule with a higher priority than the generating rules, can, in some cases, reduce the number of redundant merging combinations. Going back to the example, if the merging rule is applied already before the creation of the forth r-neighbour z, we only have three non-deterministic alternatives for the first merging step (Figure 5). After also adding z to the completion graph, a fur-

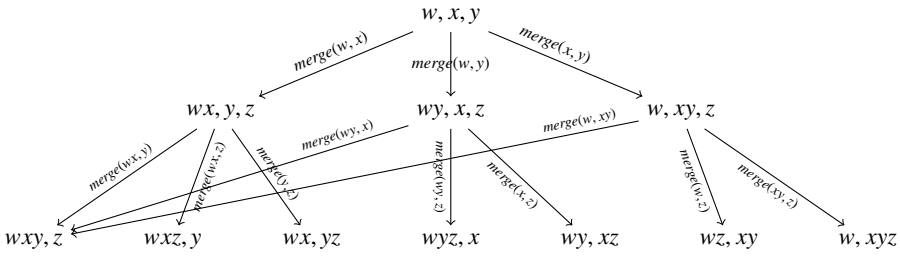

Fig. 5. Non-deterministic alternatives for pair-based merging with a prioritized $\leq$-rule

ther merging step is necessary resulting in three further possible merging pairs for each of the three alternatives. Out of the nine alternatives, there are again merging combinations that lead to the same outcome: $merge(merge(w,x),y)$, $merge(merge(w,y),x)$ and $merge(merge(x,y),w)$. Although the amount of redundant merging combinations can be reduced, it is not always possible to apply this strategy since the concept that triggers the application of the $\leq$-rule can be hidden within disjunctions or may be added via propagations over inverse roles at a later stage.

The problem is very similar to the syntactic branching search [1], where unsatisfiable concepts of non-disjoint branches might have to be evaluated multiple times. The semantic branching technique is commonly used to avoid such redundant evaluations and in the merging context an analogous approach can be very beneficial.

In order to apply this technique, all nodes of previously tested merging pairs are set to be pairwise distinct. For example, when merging (w,x) in the first merging step leads to a clash, w and x are set to be distinct because this combination has been tested and should be avoided in future tests. In the second alternative, the nodes w and y are merged, which leads to $wy\dot{\neq}x$. As a result of the inequality, $merge(merge(w,y),x)$ is never tested in the second merging step (Figure 6). If also merging w and y fails, a further inequality $w\dot{\neq}y$ is added. Finally, for the last two alternatives of the first merging step the inequality constraints prevent further merging and show that these alternatives are unsatisfiable. Summing up, with the inequalities the total number of non-deterministic alternatives can be reduced to nine in this example. Unfortunately, similarly sophisticated merging techniques can hardly be found in current reasoners.

Apart from using the inequality information, the *pool-based merging* method that we propose also prevents the redundant evaluation of previously computed merging attempts. Furthermore it works very well in combination with dependency directed backtracking due to the thin and uniform branching tree.

Regarding the implementation of the pool-based merging method, the nodes that have to be merged are managed in a queue. Each merging step takes the next node from the queue and non-deterministically inserts this node into a so-called *pool*, where the number of pools corresponds to the required cardinality. All pools are considered as distinct and nodes within one pool are merged together. If there are several empty pools, we will only generate one alternative, where the node is inserted in one of these

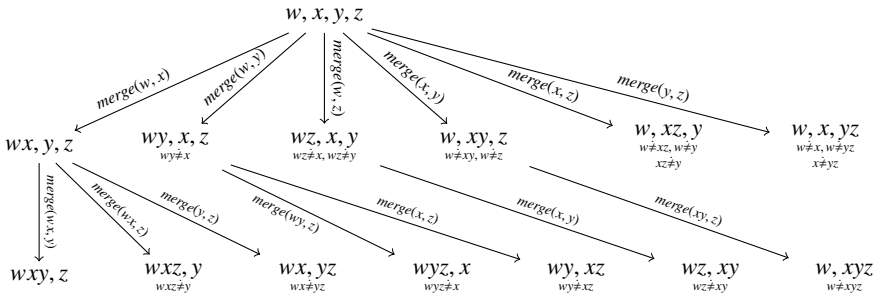

Fig. 6. Non-deterministic merging alternatives with added inequality information

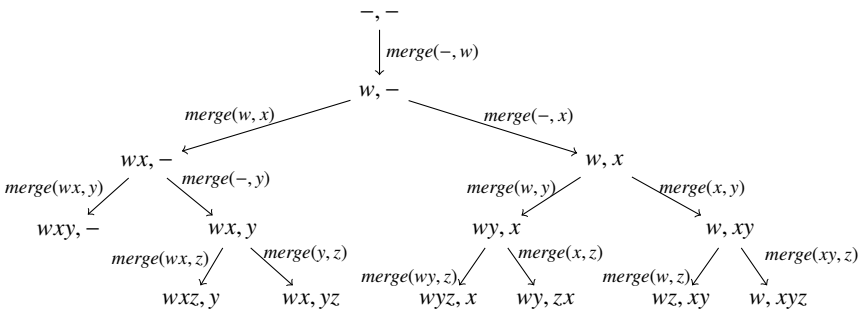

Fig. 7. Pool-based merging approach to avoid redundant evaluation of previous merging attempts

empty pools. If several empty pools were initialised with the same node, once again redundant merging combinations would have to be evaluated. For the example, the generated merging combinations due to the pool based merging procedure are illustrated in Figure 7. At the beginning, all nodes are in the queue and both pools are empty. In the first merging step the node w is taken from the queue and inserted to the first empty pool. In the second step the next node x is non-deterministically inserted into the first pool together with the node w or into another empty pool. This process continues until the cardinality restriction is satisfied. Note that z is not removed from the queue for the alternative shown on the left-hand side since the cardinality is already satisfied. If a clash occurs in an alternative, all relevant merging steps can be identified with the dependency directed backtracking. Different insertion alternatives are, therefore, only tested for nodes that are involved in the clashes. In the worst-case also the pool based merging is systematically testing all possible combinations, but the different generation of these alternatives prevents redundant evaluations. Other tableau expansions rules for $\mathcal{SROIQ}$, such as the *choose*- or the *NN*-rule, are not influenced by the merging method, consequently also qualified cardinality restrictions are supported in combination with the pool based merging.

5 Evaluation

Our Konclude reasoning system implements the enhanced optimisation techniques for $\mathcal{SROIQ}$ described above. In the following, we first compare different caching methods. Furthermore, we benchmark our pool-based merging technique against the standard pair-based approach that is used in most other systems.

We evaluate dependency directed backtracking and unsatisfiability caching with the help of concept satisfiability tests from the well-known DL 98 benchmark suite [13] and spot tests regarding cardinality restrictions and merging first proposed in [15]. From the DL 98 suite we selected satisfiable and unsatisfiable test cases (with _n resp. _p postfixes) and omitted those for which unsatisfiability caching is irrelevant and tests that were too easy to serve as meaningful and reproducible sample.

With respect to caching, we distinguish between precise caching and label caching as described in Section 3.2. To recall, precise caching stores precise cache entries con-

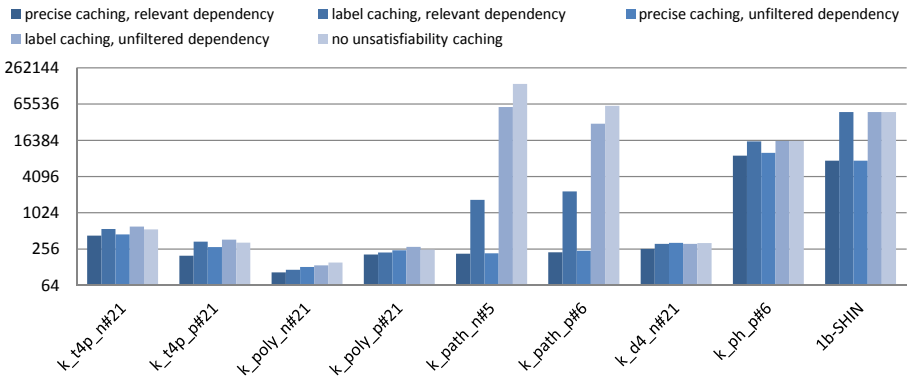

Fig. 8. Log scale comparison of processed alternatives for different caching methods

sisting of only those backtraced sets of concepts that are explicitly known to cause an unsatisfiability in combination with subset retrieval, while label caching stores and returns only entire node labels.

Independent of the caching method, we distinguish between unfiltered and relevant dependencies for further dependency backtracing after a cache hit. *Unfiltered dependency* denotes the backtracing technique that uses all the concept facts and their dependencies within a node label, for which the unsatisfiability has been found in the cache. In contrast, *relevant dependency* uses only those facts and dependencies of a node label for further backtracing that caused the unsatisfiability (as if the unsatisfiability would be found without caching).

Konclude natively maintains relevant dependencies and implements precise unsatisfiability caching based on hash data structures [11] in order to efficiently facilitate subset cache retrieval. Figure 8 shows the total number of processed non-deterministic alternatives for five configurations of caching precision and dependency handling for a selection of test cases solvable within one minute. Note that runtime is not a reasonable basis of comparison since all configuration variants of Figure 8 have been implemented (just for the purpose of evaluation) on top of the built-in and computationally more costly precise caching approach.

Figure 8 reveals that, amongst the tested configurations, precise caching provides the most effective pruning method. For some test cases it can reduce the number of non-deterministic alternatives by two orders of magnitude in comparison to label caching with unfiltered dependencies. Particularly the test cases k_path_n/p are practically solvable for Konclude only with precise caching at their largest available problem size (#21). The difference between relevant and unfiltered dependencies is less significant at least within our set of test cases. If label caching and unfiltered dependencies are used together, then, sometimes, (here for the test cases k_t4p_p/n and k_poly_p) the number of processed alternatives is even increasing in comparison to the configuration without unsatisfiability caching. The reason for this is that with the less exact unfiltered dependencies additional non-deterministic branching points are identified as relevant after a cache hit (described in Section 3.3). This adverse interaction with dependency

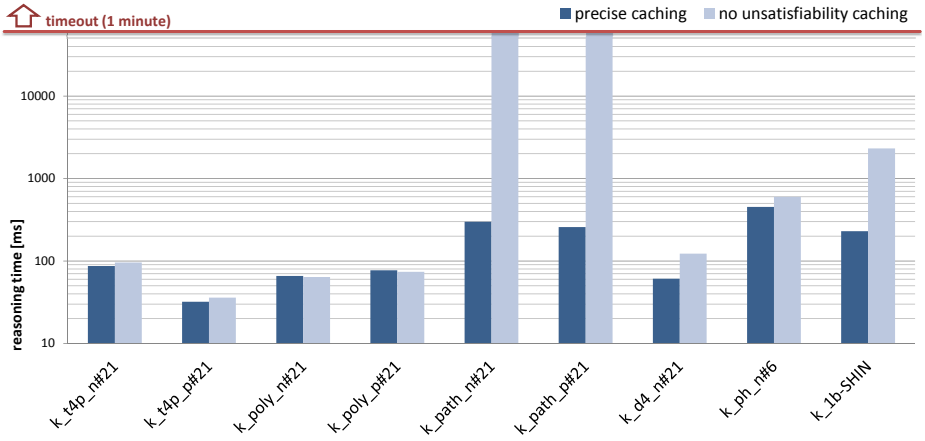

Fig. 9. Log scale comparison of reasoning time for precise and no unsatisfiability caching

directed backtracking can also occur with the precise caching and unfiltered dependency configuration. However, precise caching generally reduces the number of processed alternatives, because the precise cache entries and the subset retrieval allows that the unsatisfiability can be found earlier in the cache.

Figure 9 shows the reasoning time for precise and no unsatisfiability caching for our Konclude reasoning system. In all test cases precise caching significantly improves or at least does not downgrade overall reasoning time. For the test cases k_poly_p/n the number of processed non-deterministic alternatives can be reduced with precise caching,

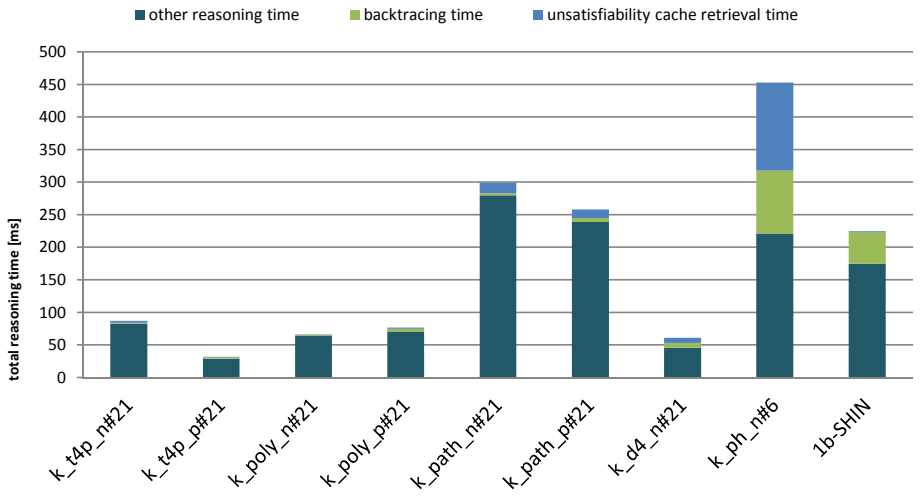

Fig. 10. Share of cache retrieval and dependency backtracing on the reasoning time

but the additional effort of managing and querying the cache prevents performance improvements. This effect can sometimes be observed, if the work in the alternatives is negligible, but the cache entries are relatively complex. Additionally, we have experienced an increase of memory usage by a worst-case factor of two in case of dependency tracking in comparison to no dependency handling.

In Figure 10 the total reasoning time is splitted into three parts: (i) the time required for the subset retrieval of the precise unsatisfiability cache, (ii) the time for dependency backtracing including the time for the creation of new cache entries, and (iii) the remaining other reasoning time. Only for the test cases k_ph_n and 1b-SHIN the times for dependency backtracing and the subset retrieval have a significant contribution to the total reasoning time. Nevertheless, with precise caching many alternatives are pruned and the reasoning time decreases. In contrast to this, traditional label caching cannot improve the reasoning time for these test cases, because the algorithm does not create exact identical node labels in the different alternatives and, therefore, the label caching has no cache hit at all.

A comparison between Konclude and other widely used reasoning systems is shown in Figure 11. Now the total processing time is shown for the test cases, i.e. the loading and pre-processing times are also included. The tests are conducted with the OWLlink interface [16] on an Intel Core i7 940 quad core processor running at 2.93 GHz. HermiT, FaCT++ and Pellet are queried with the help of the OWLlink API [5] and Konclude and RacerPro [7] are supporting OWLlink natively. The results show an average over five runs. Although Konclude can achieve good results, other reasoning systems are

[5] http://owllink-owlapi.sourceforge.net/

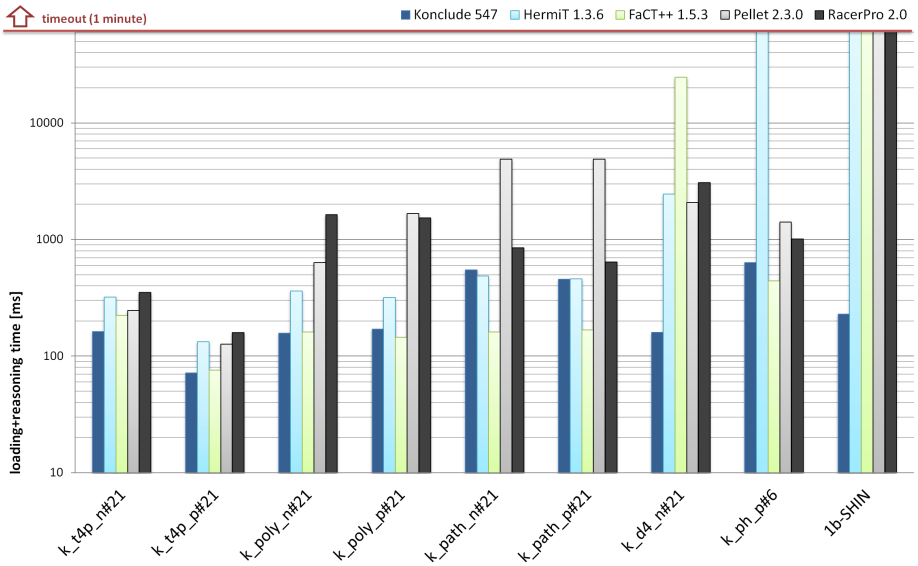

Fig. 11. Comparison of total processing time for different reasoning systems

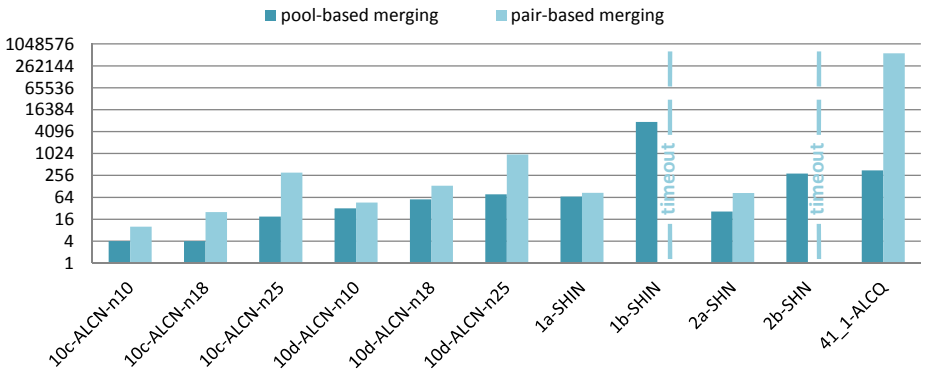

Fig. 12. Processed alternatives (on a logarithmic scale) for different merging methods

even faster for some test cases. Especially FaCT++ significantly outperforms all other reasoning systems for the k_path_p/n test cases. Without precise caching Konclude is not even able to solve these test cases within the time limit of one minute. A possible reason for this might be the different processing strategy of nodes in the completion graph or the different optimised data structures. Moreover, we have not implemented all known optimisations into Konclude so far.

Figure 12 compares pool-based with pair-based merging in terms of non-deterministic alternatives that have to be processed in order to solve selected test cases from

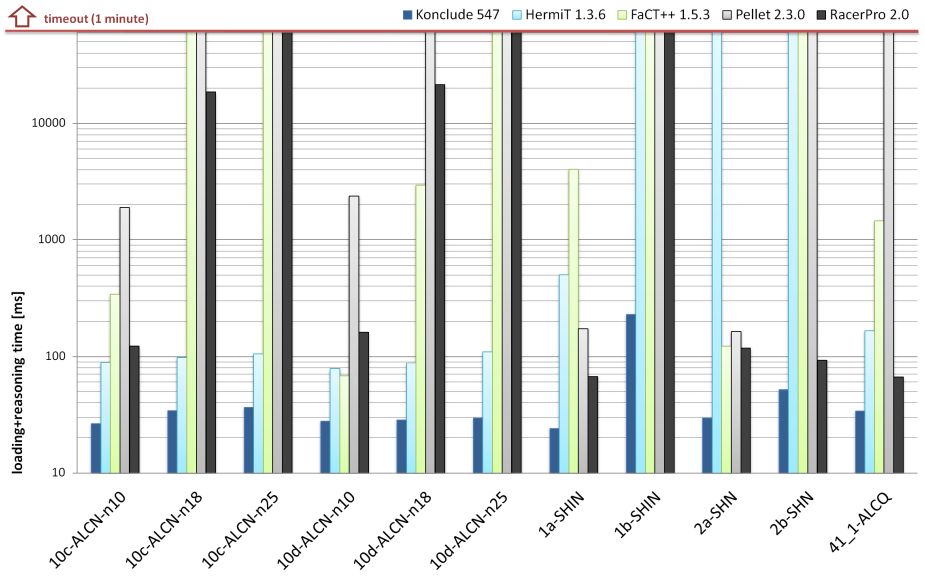

Fig. 13. Comparison of total processing time for different reasoning systems

[15]. In addition to the built-in pool-based merging we also added pair-based merging to our Konclude system. The test cases 10c and 10d are variants of the original test case 10a in terms of different problem sizes (10c) as well as more hidden contradictions nested within disjunctions (10d). The pool-based approach introduced in Sec. 4 clearly dominates the naive pair-based merging, especially when dealing with satisfiable problems (1b and 2b) and expressive DLs. Note that the test cases 1b and 2b are only solvable with pool-based merging within a one minute timeout. The required reasoning times strongly correlate to the number of processed alternatives for all test cases of Figure 12.

Advantages of the pool-based merging can also be observed, if Konclude is compared to other reasoning systems. Figure 13 shows a comparison between Konclude, HermiT, FaCT++, Pellet and RacerPro for the merging test cases, which are analogously conducted to the previous reasoner comparison. Konclude outperforms all other reasoners for all these test cases. RacerPro can use algebraic methods only for 2b-SHN and 41_1-ALCQ. However, if these test cases are extended to much greater cardinalities, then RacerPro eventually outperforms the other reasoners.

6 Conclusions

We have presented a range of optimisation techniques that can be used in conjunction with the very expressive DL $\mathcal{SROIQ}$. The presented dependency management allows for more informed backjumping, while also supporting the creation of precise cache unsatisfiability entries. In particular the precise caching approach can reduce the number of tested non-deterministic branches by up to two orders of magnitude compared to standard caching techniques. Regarding cardinality constraints, the presented pool-based merging technique also achieves a significant improvement and a number of test cases can only be solved with this optimisation within an acceptable time limit. Both techniques are well-suited for the integration into existing tableau implementations for $\mathcal{SROIQ}$ and play well with other commonly implemented optimisation techniques.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, second edn. (2007)
2. Ding, Y., Haarslev, V.: Tableau caching for description logics with inverse and transitive roles. In: Proc. 2006 Int. Workshop on Description Logics. pp. 143–149 (2006)
3. Ding, Y., Haarslev, V.: A procedure for description logic $\mathcal{ALCFI}$. In: Proc. 16th European Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'07) (2007)
4. Donini, F.M., Massacci, F.: EXPTIME tableaux for $\mathcal{ALC}$. J. of Artificial Intelligence 124(1), 87–138 (2000)
5. Faddoul, J., Farsinia, N., Haarslev, V., Möller, R.: A hybrid tableau algorithm for $\mathcal{ALCQ}$. In: Proc. 18th European Conf. on Artificial Intelligence (ECAI'08). pp. 725–726 (2008)
6. Goré, R., Widmann, F.: Sound global state caching for $\mathcal{ALC}$ with inverse roles. In: Proc. 18th European Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'09). LNCS, vol. 5607, pp. 205–219. Springer (2009)

7. Haarslev, V., Hidde, K., Möller, R., Wessel, M.: The RacerPro knowledge representation and reasoning system. Semantic Web (2012), accepted for publication
8. Haarslev, V., Möller, R.: Consistency testing: The RACE experience. In: Proceedings, Automated Reasoning with Analytic. pp. 57–61. Springer-Verlag (2000)
9. Haarslev, V., Möller, R.: High performance reasoning with very large knowledge bases: A practical case study. In: Proc. 17th Int. Joint Conf. on Artificial Intelligence (IJCAI'01). pp. 161–168. Morgan Kaufmann (2001)
10. Haarslev, V., Sebastiani, R., Vescovi, M.: Automated reasoning in $\mathcal{ALCQ}$ via SMT. In: Proc. 23rd Int. Conf. on Automated Deduction (CADE'11). pp. 283–298. Springer (2011)
11. Hoffmann, J., Koehler, J.: A new method to index and query sets. In: Proc. 16th Int. Conf. on Artificial Intelligence (IJCAI'99). pp. 462–467. Morgan Kaufmann (1999)
12. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06). pp. 57–67. AAAI Press (2006)
13. Horrocks, I., Patel-Schneider, P.F.: DL systems comparison. In: Proc. 1998 Int. Workshop on Description Logics (DL'98). vol. 11, pp. 55–57 (1998)
14. Horrocks, I., Patel-Schneider, P.F.: Optimizing description logic subsumption. J. of Logic and Computation 9(3), 267–293 (1999)
15. Liebig, T.: Reasoning with OWL – system support and insights –. Tech. Rep. TR-2006-04, Ulm University, Ulm, Germany (September 2006)
16. Liebig, T., Luther, M., Noppens, O., Wessel, M.: Owllink. Semantic Web – Interoperability, Usability, Applicability 2(1), 23–32 (2011)
17. Liebig, T., Steigmiller, A., Noppens, O.: Scalability via parallelization of OWL reasoning. In: Proc. Workshop on New Forms of Reasoning for the Semantic Web: Scalable & Dynamic (NeFoRS'10) (2010)
18. OWL Working Group, W.: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation (27 October 2009), available at `http://www.w3.org/TR/owl2-overview/`
19. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. of Web Semantics 5(2), 51–53 (2007)
20. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Proc. 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06). LNCS, vol. 4130, pp. 292–297. Springer (2006)
21. Tsarkov, D., Horrocks, I., Patel-Schneider, P.F.: Optimizing terminological reasoning for expressive description logics. J. of Automated Reasoning 39, 277–316 (2007)

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

*91-02** *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

*91-03** *Alfons Geser*
Relative Termination

*91-04** *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

*91-06** *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

*91-07** *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

*91-08** *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

*92-01** *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

*92-02** *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

*92-04** *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

*92-05** *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

*92-06** *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

*92-07** *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

*92-08** *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

*92-09** *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

*92-14** *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars:
Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman,Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-11 *Thomas Beuter, Peter Dadam*:
Prinzipien der Replikationskontrolle in verteilten Systemen

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12 *Gerhard Schellhorn*
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

98-13 *Gerhard Schellhorn, Wolfgang Reif*
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

98-14 *Mohammad Ali Livani*
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

98-15 *Mohammad Ali Livani, Jörg Kaiser*
Predictable Atomic Multicast in the Controller Area Network (CAN)

99-01 *Susanne Boll, Wolfgang Klas, Utz Westermann*
A Comparison of Multimedia Document Models Concerning Advanced Requirements

99-02 *Thomas Bauer, Peter Dadam*
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

99-03 *Uwe Schöning*
On the Complexity of Constraint Satisfaction

99-04 *Ercument Canver*
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

99-05 *Johannes Köbler, Wolfgang Lindner, Rainer Schuler*
Derandomizing RP if Boolean Circuits are not Learnable

99-06 *Utz Westermann, Wolfgang Klas*
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

99-07 *Peter Dadam, Manfred Reichert*
Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik ’99

99-08 *Vikraman Arvind, Johannes Köbler*
Graph Isomorphism is Low for ZPP^{NP} and other Lowness results

99-09 *Thomas Bauer, Peter Dadam*
Efficient Distributed Workflow Management Based on Variable Server Assignments

2000-02 *Thomas Bauer, Peter Dadam*
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

2000-03 *Gregory Baratoff, Christian Toepfer, Heiko Neumann*
Combined space-variant maps for optical flow based navigation

2000-04 *Wolfgang Gehring*
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05 *Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel*
Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06 *Wolfgang Reif, Gerhard Schellhorn, Andreas Thums*
Fehlersuche in Formalen Spezifikationen

2000-07 *Gerhard Schellhorn, Wolfgang Reif (eds.)*
FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification

2000-08 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09 *Thomas Bauer, Peter Dadam*
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Adaptives und verteiltes Workflow-Management

2000-11 *Christian Heinlein*
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01 *Hubert Hug, Rainer Schuler*
DNA-based parallel computation of simple arithmetic

2001-02 *Friedhelm Schwenker, Hans A. Kestler, Günther Palm*
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03 *Hans A. Kestler, Friedhelm Schwenker, Günther Palm*
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04 *Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm*
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02 *Walter Guttmann*
Deriving an Applicative Heapsort Algorithm

2002-03 *Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk*
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01 *Manfred Reichert, Stefanie Rinderle, Peter Dadam*
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
On Dealing With Semantically Conflicting Business Process Changes.

2003-05 *Christian Heinlein*
Dynamic Class Methods in Java

2003-06 *Christian Heinlein*
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

2003-07 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values
(Corrected Version)

2003-08 *Changling Liu, Jörg Kaiser*
Survey of Mobile Ad Hoc Network Routing Protocols)

2004-01 *Thom Frühwirth, Marc Meister (eds.)*
First Workshop on Constraint Handling Rules

2004-02 *Christian Heinlein*
Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

2004-03 *Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)*
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

2005-01 *Armin Wolf, Thom Frühwirth, Marc Meister (eds.)*
19th Workshop on (Constraint) Logic Programming

2005-02 *Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven*
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

2005-03 *Walter Guttmann, Markus Maucher*
Constrained Ordering

2006-01 *Stefan Sarstedt*
Model-Driven Development with ACTIVECHARTS, Tutorial

2006-02 *Alexander Raschke, Ramin Tavakoli Kolagari*
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

2006-03 *Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari*
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

2006-04 *Thorsten Liebig*
Reasoning with OWL - System Support and Insights –

2008-01 *H.A. Kestler, J. Messner, A. Müller, R. Schuler*
On the complexity of intersecting multiple circles for graphical display

2008-02 *Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer*
Architectural Design of Flexible Process Management Technology

2008-03 *Frank Raiser*
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04 *Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander*
Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05 *Markus Kalb, Claudia Dittrich, Peter Dadam*
Support of Relationships Among Moving Objects on Networks

2008-06 *Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)*
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07 *M. Maucher, U. Schöning, H.A. Kestler*
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

2008-08 *Henning Wunderlich*
Covers have structure

2008-09 *Karl-Heinz Niggl, Henning Wunderlich*
Implicit characterization of FPTIME and NC revisited

2008-10 *Henning Wunderlich*
On span-P^{cc} and related classes in structural communication complexity

2008-11 *M. Maucher, U. Schöning, H.A. Kestler*
On the different notions of pseudorandomness

2008-12 *Henning Wunderlich*
On Toda's Theorem in structural communication complexity

2008-13 *Manfred Reichert, Peter Dadam*
Realizing Adaptive Process-aware Information Systems with ADEPT2

2009-01 *Peter Dadam, Manfred Reichert*
The ADEPT Project: A Decade of Research and Development for Robust and Fexible Process Support
Challenges and Achievements

2009-02 *Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch*
Von ADEPT zur AristaFlow® BPM Suite – Eine Vision wird Realität "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen

2009-03 *Alena Hallerbach, Thomas Bauer, Manfred Reichert*
Correct Configuration of Process Variants in Provop

2009-04 *Martin Bader*
On Reversal and Transposition Medians

2009-05 *Barbara Weber, Andreas Lanz, Manfred Reichert*
Time Patterns for Process-aware Information Systems: A Pattern-based Analysis

2009-06 *Stefanie Rinderle-Ma, Manfred Reichert*
Adjustment Strategies for Non-Compliant Process Instances

2009-07 *H.A. Kestler, B. Lausen, H. Binder H.-P. Klenk. F. Leisch, M. Schmid*
Statistical Computing 2009 – Abstracts der 41. Arbeitstagung

2009-08 *Ulrich Kreher, Manfred Reichert, Stefanie Rinderle-Ma, Peter Dadam*
Effiziente Repräsentation von Vorlagen- und Instanzdaten in Prozess-Management-Systemen

2009-09 *Dammertz, Holger, Alexander Keller, Hendrik P.A. Lensch*
Progressive Point-Light-Based Global Illumination

2009-10 *Dao Zhou, Christoph Müssel, Ludwig Lausser, Martin Hopfensitz, Michael Kühl, Hans A. Kestler*
Boolean networks for modeling and analysis of gene regulation

2009-11 *J. Hanika, H.P.A. Lensch, A. Keller*
Two-Level Ray Tracing with Recordering for Highly Complex Scenes

2009-12 *Stephan Buchwald, Thomas Bauer, Manfred Reichert*
Durchgängige Modellierung von Geschäftsprozessen durch Einführung eines Abbildungsmodells: Ansätze, Konzepte, Notationen

2010-01 *Hariolf Betz, Frank Raiser, Thom Frühwirth*
A Complete and Terminating Execution Model for Constraint Handling Rules

2010-02 *Ulrich Kreher, Manfred Reichert*
Speichereffiziente Repräsentation instanzspezifischer Änderungen in Prozess-Management-Systemen

2010-03 *Patrick Frey*
Case Study: Engine Control Application

2010-04 *Matthias Lohrmann und Manfred Reichert*
Basic Considerations on Business Process Quality

2010-05 *HA Kestler, H Binder, B Lausen, H-P Klenk, M Schmid, F Leisch (eds):*
Statistical Computing 2010 - Abstracts der 42. Arbeitstagung

2010-06 *Vera Künzle, Barbara Weber, Manfred Reichert*
Object-aware Business Processes: Properties, Requirements, Existing Approaches

2011-01 *Stephan Buchwald, Thomas Bauer, Manfred Reichert*
Flexibilisierung Service-orientierter Architekturen

2011-02 *Johannes Hanika, Holger Dammertz, Hendrik Lensch*
Edge-Optimized À-Trous Wavelets for Local Contrast Enhancement with Robust Denoising

2011-03 *Stefanie Kaiser, Manfred Reichert*
Datenflussvarianten in Prozessmodellen: Szenarien, Herausforderungen, Ansätze

2011-04 *Hans A. Kestler, Harald Binder, Matthias Schmid, Friedrich Leisch, Johann M. Kraus (eds):*
Statistical Computing 2011 - Abstracts der 43. Arbeitstagung

2011-05 *Vera Künzle, Manfred Reichert*
PHILharmonicFlows: Research and Design Methodology

2011-06 *David Knuplesch, Manfred Reichert*
Ensuring Business Process Compliance Along the Process Life Cycle

2011-07 *Marcel Dausend*
Towards a UML Profile on Formal Semantics for Modeling Multimodal Interactive Systems

2011-08 *Dominik Gessenharter*
Model-Driven Software Development with ACTIVECHARTS - A Case Study

2012-01 *Andreas Steigmiller, Thorsten Liebig, Birte Glimm*
Extended Caching, Backjumping and Merging for Expressive Description Logics

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm