Optimizing Reasoning with Qualified Number Restrictions in SHQ

N. Farsiniamarj and V. Haarslev
Concordia University, Montreal, Canada

1 Introduction

Using $\mathcal{SHQ}$ one can express number restrictions on role fillers of individuals. A typical question for a DL reasoner would be whether the concept $\forall hasCredit.(Science \sqcup Engineering \sqcup Business) \sqcap (\geq 120\, hasCredit.(Science \sqcup Engineering)) \sqcap (\geq 140\, hasCredit) \sqcap (\leq 32\, hasCredit.(Science \sqcup Business)) \sqcap (\leq 91\, hasCredit.Engineering)$ is satisfiable. Most DL tableau algorithms [9, 1, 12] test the satisfiability of such a concept by first satisfying all at-least restrictions, e.g., by creating 260 $hasCredit$-fillers, of which 120 are instances of $(Science \sqcup Engineering)$. Eventually, a nondeterministic choose-rule assigns to each of these 260 individuals $(Science \sqcup Business)$ or $\neg(Science \sqcup Business)$, and $Engineering$ or $\neg Engineering$. In case an at-most restriction is violated, e.g., a student has more than 91 $hasCredit$-fillers of $Engineering$, a nondeterministic merge-rule tries to reduce the number of these individuals by merging a pair of individuals until the upper bound specified in this at-most restriction is satisfied. Searching for a model in such an arithmetically uninformed or blind way is usually very inefficient.

Our hybrid calculus is based on a standard tableau for $\mathcal{SH}$ [1] modified and extended to deal with qualified number restrictions and works with an inequation solver based on integer linear programming. The tableau rules encode number restrictions into a set of inequations using the so-called atomic decomposition technique [16]. The set of inequations is processed by the inequation solver which finds, if possible, a minimal non-negative integer solution (distribution of role fillers constrained by number restrictions) satisfying the inequations. The tableau rules ensure that such a distribution of role fillers also satisfies the logical restrictions.

Since this hybrid algorithm collects all the information about arithmetic expressions before creating any role filler, it will not satisfy any at-least restriction by violating an at-most restriction and there is no need for a mechanism of merging role fillers and its performance is not affected by the values of numbers occurring in number restrictions. Since the solution from the inequation solver satisfies all numerical restrictions imposed by at-least and at-most restrictions, our calculus needs to create only one so-called *proxy individual* (inspired by [6]) representing a set of role fillers. Considering all these features the proposed hybrid algorithm is well suited to improve average case performance. Furthermore, in [2, Chapter 6] it has been shown that a tableau procedure extended by global caching and an algebraic method similar to the one presented in this paper and in [16, 8] is worst-case optimal for $\mathcal{SHIQ}$. Although the calculus presented in our paper is different from the one in [2] we conjecture that the result presented in [2] can be transferred to our work because the use of integer linear programming was essential to proving worst-case optimality.

This calculus extends our work presented in [3] by (i) covering Abox consistency for $\mathcal{SHQ}$ with general Tboxes, and (ii) introducing the use of proxy individuals representing sets of role fillers. It complements the work in [4] (which extends the work

Table 1. Syntax and semantics of $\mathcal{SHQ}$

Syntax	Semantics
A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}, A \in N_C$
$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\forall R.C$	$\{x \mid \forall y : (x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$
$\geq n\, R.C$	$\{x \mid \#R^{\mathcal{I}}(x,C) \geq n\}$
$\leq m\, R.C$	$\{x \mid \#R^{\mathcal{I}}(x,C) \leq m\}$

Syntax	Semantics
$R \in N_R$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
$R \in N_{RT}$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{+}$
$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
$a : C$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
$(a,b) : R$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
$a \neq b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$

in [3] to $\mathcal{ALCOQ}$) by additionally dealing with role hierarchies and transitive roles. This paper also discusses practical aspects of using the hybrid method and presents evaluation results based on an implemented prototype. It also significantly differs from the work presented in [15, 16] where no formal calculus was proposed and the covered logic did not support Abox consistency for $\mathcal{SHQ}$ with general Tboxes. Our early work presented in [8] did not deal with Aboxes and was based on a recursive calculus without using proxy individuals and did not provide any proof for soundness, completeness, or termination.

2 Preliminaries

2.1 Description Logic $\mathcal{SHQ}$

In the following, three disjoint sets are defined; N_C is the set of concepts names; $N_R = N_{RT} \cup N_{RS}$ is the set of all role names which consists of transitive (N_{RT}) and non-transitive (N_{RS}) roles; I is the set of all individuals, while $I_{\mathcal{A}} \subseteq I$ is the set of individuals occurring in an Abox $\mathcal{A}$. A role is called *simple* if it is neither transitive nor has any transitive sub-role. In order to remain decidable qualified number restrictions are only allowed for simple roles (except for concepts of the form $\geq 1\, R.C$). In Table 1, we use a standard Tarski-style semantics $\mathcal{I}$ and assume that C, D are arbitrary concept descriptions, $R, S \in N_R$, $a, b \in I_{\mathcal{A}}$, and $\#R^{\mathcal{I}}(x, C)$ denotes the cardinality of $\{y \mid (x,y) \in R^{\mathcal{I}}, y \in C^{\mathcal{I}}\}$. Let $\sqsubseteq_*$ be the transitive, reflexive closure of $\sqsubseteq$ over N_R. A concept description C is said to be satisfiable by an interpretation $\mathcal{I}$ iff $C^{\mathcal{I}} \neq \emptyset$. A role hierarchy $\mathcal{R}$ is a set of assertions of the form $R \sqsubseteq S$ where $R, S \in N_R$.

A $\mathcal{SHQ}$ Tbox $\mathcal{T}$ is a finite set of axioms of the form $C \sqsubseteq D$ or $C \equiv D$ where C, D are concept expressions and $C \equiv D$ is the placeholder for $\{C \sqsubseteq D,\, D \sqsubseteq C\}$. A Tbox $\mathcal{T}$ is satisfiable by an interpretation $\mathcal{I}$ iff $\mathcal{I}$ satisfies all axioms in $\mathcal{T}$ and $\mathcal{R}$. A $\mathcal{SHQ}$ Abox $\mathcal{A}$ w.r.t. a Tbox $\mathcal{T}$ is a finite set of assertions of the form $a : C$, $(a,b) : R$, and $a \neq b$. An Abox $\mathcal{A}$ is satisfiable by an interpretation $\mathcal{I}$ iff $\mathcal{I}$ satisfies $\mathcal{T}$ and $\mathcal{R}$ and all assertions in $\mathcal{A}$. Let $a, b \in I$ be (possibly unnamed) individuals, given an assertion $(a,b) : R$, a is called a predecessor of b and b a successor or role filler of a. The R-fillers of a are defined as $Fil(a, R) = \{b \mid (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}\}$.

2.2 Reducing $\mathcal{SHQ}$ to $\mathcal{SHN}^{\backslash}$

Inspired by [16] we transform given qualified number restrictions to unqualified number restrictions and add a new role-set difference operator $\forall(R \backslash R').C$ with $(\forall(R \backslash R').C)^{\mathcal{I}} = \{x \mid \forall y : (x,y) \in (R^{\mathcal{I}} \setminus R'^{\mathcal{I}}) \Rightarrow y \in C^{\mathcal{I}}\}$. The resulting language is called $\mathcal{SHN}^{\backslash}$. Let $\dot{\neg} C$ denote the standard negation normal form (NNF) of $\neg C$. We define a recursive

function unQ which rewrites a $\mathcal{SHQ}$ concept description into $\mathcal{SHN}^{\setminus}$. It is important to note that this rewriting process always introduces for each transformed qualified number restriction a unique new role.

The function unQ transforms the input description into its NNF and replaces qualified number restrictions. In the following each R' is a new role in N_R with $\mathcal{R} := \mathcal{R} \cup \{R' \sqsubseteq R\}$: $unQ(C) := C$ if $C \in N_C$; $unQ(\neg C) := \neg C$ if $C \in N_C$, $unQ(\doteq C)$ otherwise; $unQ(\forall R.C) := \forall R.unQ(C)$; $unQ(C \sqcap D) := unQ(C) \sqcap unQ(D)$; $unQ(C \sqcup D) := unQ(C) \sqcup unQ(D)$; $unQ(\geq nR.C) := (\geq nR') \sqcap \forall R'.unQ(C)$; $unQ(\leq nR.C) := (\leq nR') \sqcap \forall (R \backslash R').unQ(\doteq C)$; $unQ(a\!:\!C) := a\!:\!unQ(C)$.

It is easy to show that the concept $(\leq n\, R.C)$ is equisatisfiable with the expression $(\forall (R \backslash R').\neg C \sqcap \leq n\, R')$ with R' fresh in $\mathcal{R}$ and $R' \sqsubseteq R$ (see [5] for the proof). Since unQ introduces a negation itself, the negated description needs to be converted to NNF before further applying unQ. Our language is not closed under negation w.r.t. the concept descriptions created by unQ. However, our calculus ensures that these concept descriptions will never be negated.

3 A Hybrid Abox Tableau Calculus for $\mathcal{SHQ}$

3.1 Preprocessing

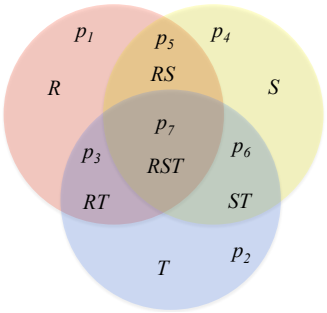
As a preprocessing step, the input Abox and Tbox are transformed into $\mathcal{SHN}^{\setminus}$ as defined in Section 2.2. Similar to [13], in order to propagate Tbox axioms to all individuals we define the concept $C_{\mathcal{T}} := \bigsqcap_{C \sqsubseteq D \in \mathcal{T}} unQ(\doteq C \sqcup D)$ and U as a new transitive role in the role hierarchy $\mathcal{R}$ extended by $\{R \sqsubseteq U \mid R \in N_R\}$. In order to examine the consistency of $\mathcal{A}$, the algorithm first extends $\mathcal{A}$ by $\{x_0\!:\!(C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}})\} \cup \{(x_0, x)\!:\!U \mid x \text{ occurs in } \mathcal{A}\}$, where x_0 is new in $\mathcal{A}$. By this means, we impose the axioms in the Tbox on all individuals in $\mathcal{A}$. Furthermore, we rewrite the role assertions in $\mathcal{A}$ as follows.

If $x, y \in I_{\mathcal{A}}$ then there exists a unique role name $R_{xy} \in N_R$ only used to represent that y is an R-filler of x with $R_{xy} \sqsubseteq R \in \mathcal{R}$. In other words whenever $(x, z) : R_{xy}$ we have $y^{\mathcal{I}} = z^{\mathcal{I}}$. We replace role assertions of the form $(b, c) : R$ by $b : (\geq 1\, R_{bc} \sqcap \leq 1\, R_{bc})$ because they impose a numerical restriction on individuals. For example, an assertion $(b, c)\!:\!R$ implies there exists exactly one R_{bc}-filler for b which is c.

3.2 Completion Forest

The closure $clos(E)$ of a concept expression E is the smallest set of concepts such that: $E \in clos(E)$, $(\neg D) \in clos(E) \Rightarrow D \in clos(E)$, $(C \sqcup D) \in clos(E)$ or $(C \sqcap D) \in clos(E) \Rightarrow C \in clos(E)$ and $D \in clos(E)$, $(\forall R.C) \in clos(E) \Rightarrow C \in clos(E)$, $(\geq n\, R.C) \in clos(E)$ or $(\leq m\, R.C) \in clos(E) \Rightarrow C \in clos(E)$. For a Tbox $\mathcal{T}$ we define $clos(\mathcal{T})$ such that if $(C \sqsubseteq D) \in \mathcal{T}$ or $(C \equiv D) \in \mathcal{T}$ then $clos(C) \subseteq clos(\mathcal{T})$ and $clos(D) \subseteq \mathcal{T}$. Similarly for an Abox $\mathcal{A}$ we define $clos(\mathcal{A})$ such that if $(a\!:\!C) \in \mathcal{A}$ then $clos(C) \subseteq clos(\mathcal{A})$.

A *completion forest* $\mathcal{F} = (V, E, \mathcal{L}, \mathcal{L}_E)$ for a $\mathcal{SHQ}$ Abox $\mathcal{A}$ is composed of a set of arbitrarily connected nodes as the roots of completion trees (if one ignores the connections between root nodes). Every node $x \in V$ is labeled by $\mathcal{L}(x) \subseteq clos(\mathcal{A})$ and $\mathcal{L}_E(x)$ as a set of inequations of the form $(\sum_{i \in 1..k} v_i) \bowtie n$ with $\bowtie \in \{\leq, \geq\}$, $n \in \mathbb{N}$, and $v_i \in \mathcal{V}$ where $\mathcal{V}$ is a set of variables (see also below); each edge $\langle x, y \rangle \in E$ is labeled by the set $\mathcal{L}(\langle x, y \rangle) \subseteq N_R$, and x (y) is called a predecessor (successor or role filler) of y (x), and the transitively closed set of successors (predecessors) is called



$$p_1 = \{R\},\ p_2 = \{T\},\ p_4 = \{S\},$$
$$p_3 = \{R,T\},\ p_5 = \{R,S\},\ p_6 = \{S,T\},$$
$$p_7 = \{R,S,T\}$$
$$\alpha(v_{001}) = p_1,\ \alpha(v_{010}) = p_2,\ \alpha(v_{100}) = p_4,$$
$$\alpha(v_{011}) = p_3,\ \alpha(v_{101}) = p_5,\ \alpha(v_{110}) = p_6,$$
$$\alpha(v_{111}) = p_7$$
$$\mathcal{L}_E(x) = \left\{ \begin{array}{l} v_{001} + v_{011} + v_{101} + v_{111} \geq 3, \\ v_{010} + v_{011} + v_{110} + v_{111} \leq 2, \\ v_{100} + v_{101} + v_{110} + v_{111} \leq 1, \\ v_{100} + v_{101} + v_{110} + v_{111} \geq 1 \end{array} \right\}$$

Fig. 1. Atomic Decomposition Example

descendants (ancestors) respectively. We maintain the distinction between nodes of a forest by the relation $\doteq$.

The set R_x of *related roles* for a node x is defined as $R_x = \{S \mid \{\leq n\, S,\ \geq m\, S\} \cap \mathcal{L}(x) \neq \emptyset\}$. We define a *partitioning* $\mathcal{RS}_x = (\bigcup_{RS \subseteq R_x} \{RS\}) \setminus \emptyset$ and for a partition $RS_x \in \mathcal{RS}_x$ we define $RS_x^{\mathcal{I}} = (\bigcap_{S \in RS_x} Fil^{\mathcal{I}}(x,S)) \setminus (\bigcup_{S \in R_x \setminus RS_x} Fil^{\mathcal{I}}(x,S))$ such that $Fil^{\mathcal{I}}(x,S) = \{y^{\mathcal{I}} \mid y \in Fil(x,S)\}$. $RS_x^{\mathcal{I}}$ represents the interpretation of fillers of x that are fillers for the roles in RS but not fillers for the roles in $R_x \setminus RS$. Therefore, by definition the fillers of x associated with the partitions in $\mathcal{RS}_x$ are mutually disjoint w.r.t. the interpretation $\mathcal{I}$.

We assume a set $\mathcal{V}$ of variables and by using a mapping $\alpha : \mathcal{V} \leftrightarrow \mathcal{RS}_x$ we associate a unique variable $v \in \mathcal{V}$ with each partition RS_x in $\mathcal{RS}_x$ such that $\alpha(v) = RS_x$. Let $\mathcal{V}^R$ be defined as the set of all variables related to a role R such that $\mathcal{V}^R = \{v \in \mathcal{V} \mid R \in \alpha(v)\}$. A function ξ is used to map number restrictions to inequations of the form $\xi(R, \bowtie, n) := (\sum_{v_i \in \mathcal{V}^R} v_i) \bowtie n$ where $\bowtie \in \{\leq, \geq\}$ and $n \in \mathbb{N}$.

Since all concept restrictions for node successors that are not root nodes are propagated through universal restrictions for roles and a new role is created for each role filler occurring in a role assertion, we can conclude that all nodes in a certain node partition share the same restrictions and can be dealt with as a unit. We call this unit *proxy node* which is a representative of possibly more than one node. We define the mapping $card : V \rightarrow \mathbb{N}$ to indicate the cardinality associated with proxy nodes.

A node x in a forest F is blocked by a node y iff x, y are not root nodes and y is an ancestor of x such that $\mathcal{L}(x) \subseteq \mathcal{L}(y)$. A node x contains a clash iff there exists a concept name $A \in N_C$ such that $\{A, \neg A\} \subseteq \mathcal{L}(x)$ (logical clash) or $\mathcal{L}_E(x)$ does not have a non-negative integer solution (arithmetic clash).

An arithmetic solution is represented using the function $\sigma : \mathcal{V} \rightarrow \mathbb{N}$ which assigns a non-negative integer value to each variable in $\mathcal{V}$. Let $\mathcal{V}_x$ be the set of variables assigned to an individual x, $\mathcal{V}_x = \{v \in \mathcal{V} \mid v \text{ occurs in } \mathcal{L}_E(x)\}$. We define a set of solutions Ω for x as $\Omega(x) := \{\sigma(v) = n \mid\ n \in \mathbb{N}, v \in \mathcal{V}_x\}$. Notice that the goal function of the inequation-solver is to minimize the sum of the variables occurring in the input inequations.

The algorithm starts with the forest $\mathcal{F}_{\mathcal{A}}$ with the individuals mentioned in $\mathcal{A}$ as root nodes. For each $a \in I_{\mathcal{A}}$ a root node x_a will be created with $\mathcal{L}(x_a) = \{C \mid (a:C) \in \mathcal{A}\}$. Additionally, for every root node x we set $card(x) = 1$.

We illustrate these definitions with a simple example shown in Figure 1. Assume for a node x that we have $\mathcal{L}(x) = \{\geq 3\, R, \leq 2\, T, \geq 1\, S, \leq 1\, S\}$. Therefore, we have $R_x = \{R, T, S\}$ and $\mathcal{RS}_x$ consists of 7 different partitions named $p_1, \ldots, p_7$ (see Figure 1). Assuming a binary coding of the indices of variables, where the first digit from the right represents R, the second T, and the last S, we define the variables such that each v_i is associated with its corresponding partition (see $\alpha(v)$ in Figure 1). Hence, the number restrictions in $\mathcal{L}(x)$ can be translated to the set of inequations in $\mathcal{L}_E(x)$ as shown in the bottom-right part of Figure 1.

3.3 Completion Rules

The completion rules are shown in Figure 2. The completion rules are always applied with the following priorities (given in decreasing order). A rule of lower priority can never be applied if another one with a higher priority is applicable. 1. *reset*-Rule and *merge*-Rule. 2. $\sqcap$-Rule, $\sqcup$-Rule, $\forall$-Rule, $\forall_{\setminus}$-Rule, $\forall_{+}$-Rule, ch-Rule, *disjoint*-Rule, *equal*-Rule, and *hierarchy*-Rule 3. $\leq$-Rule and $\geq$-Rule. 4. *fil*-Rule. There are two limitations on the application of the rules: (i) priority of the rules, (ii) rules are only applicable to nodes that are not blocked.

A completion forest $\mathcal{F}$ is called *complete* if no completion rule is applicable to any node of $\mathcal{F}$. A completion forest $\mathcal{F}$ is called *clash-free* if none of the clash triggers is applicable to any node of $\mathcal{F}$.

The $\sqcap$-Rule, $\sqcup$-Rule, $\forall$-Rule, and the $\forall_{+}$-Rule are similar to the ones in standard tableau algorithms. The $\forall_{\setminus}$-Rule implements the universal restriction based on role set difference. *$\leq$-Rule, $\geq$-Rule:* The $\leq$-Rule and the $\geq$-Rule translate the number restrictions, based on the atomic decomposition technique, into inequations and add them to $\mathcal{L}_E(x)$.

ch-Rule: The intuition behind the *ch*-Rule is due to the partitioning consequences. When we partition all successors for a node, we actually consider all possible cases for the role successors. If a partition p_x for a node x is logically unsatisfiable, the corresponding variable v with $\alpha(v) = p_x$ should be zero. But if it is logically satisfiable, nothing but the current set of inequations can restrict the number of nodes being fillers for the roles in the partition p_x.

The *disjoint*-Rule preserves the semantics of Abox assertions of the form $a \neq b$. The *equal*-Rule ensures the equivalence of the R_{xy}- and S_{xy}-fillers. The *hierarchy*-Rule enforces the semantics of role hierarchies on partition variables. The *merge*-Rule merges two successors (and root nodes) z_a, z_b of a root node z_x into one node by replacing every occurrence of z_a in the completion graph by z_b. The labels $\mathcal{L}$ and $\mathcal{L}_E$ of z_b are unified with the corresponding labels of z_a. All incoming (outgoing) edges of z_a become now incoming (outgoing) edges of z_b.

fil-Rule: The *fil*-Rule, which has the lowest priority, is the only generating rule. It creates one successor (proxy node) for a node z_x that is not blocked and sets the cardinality of the proxy node to $\sigma(v)$. If z_x is a root node and has an already existing successor x_b, which is a also a root node, and the role set for $\langle z_x, z_b \rangle$ is empty, i.e., it has not yet been initialized or has been reset by the *reset*-rule, the role set is set to $\alpha(v)$.

Due to possible cycles between root nodes, one has to consider incremental partitioning for root nodes because a previously unknown number restriction could be propagated to a root node. Whenever $(\leq n\, R)$ or $(\geq\, m\, R)$ is added to $\mathcal{L}(x)$, the following

***reset*-Rule**	**if** x is a root node, $\{(\leq nR), (\geq nR)\} \cap \mathcal{L}(x) \neq \emptyset$, and $\forall v \in V_x : R \notin \alpha(v)$ **then** set $\mathcal{L}_E(x) := \emptyset$ and for every successor y of x set $\mathcal{L}(\langle x, y\rangle) := \emptyset$
***merge*-Rule**	**if** there exists root nodes z_x, z_a, z_b for $x, a, b \in I_\mathcal{A}$ such that $R_{xa} \in \mathcal{L}(\langle z_x, z_b\rangle)$ **then** merge the nodes z_a, z_b and their labels and replace every occurrence of z_a in the completion graph by z_b
$\sqcap$**-Rule**	**if** $(C_1 \sqcap C_2) \in \mathcal{L}(x)$ and $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$**-Rule**	**if** $(C_1 \sqcup C_2) \in \mathcal{L}(x)$ and $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{X\}$ with $X \in \{C_1, C_2\}$
$\forall$**-Rule**	**if** $\forall R.C \in \mathcal{L}(x)$ and there exists a y and R' with $R' \in \mathcal{L}(\langle x, y\rangle)$, $C \notin \mathcal{L}(y)$, $R' \sqsubseteq_* R$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\forall_\setminus$**-Rule**	**if** $\forall R \setminus S.C \in \mathcal{L}(x)$ and there exists a y and R' with $R' \in \mathcal{L}(\langle x, y\rangle)$, $R' \sqsubseteq_* R$, $C \notin \mathcal{L}(y)$, and there exists no S' such that $S' \sqsubseteq_* S$ and $S' \in \mathcal{L}(\langle x, y\rangle)$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\forall_+$**-Rule**	**if** $\forall R.C \in \mathcal{L}(x)$ and there exists a y and R', S with $R' \in \mathcal{L}(\langle x, y\rangle)$, $R' \sqsubseteq_* S$, $S \sqsubseteq_* R$, $S \in N_{RT}$, and $\forall S.C \notin \mathcal{L}(y)$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall S.C\}$
***ch*-Rule**	**If** there occurs v in $\mathcal{L}_E(x)$ with $\{v \geq 1,\ v \leq 0\} \cap \mathcal{L}_E(x) = \emptyset$ **then** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{X\}$ with $X \in \{v \geq 1,\ v \leq 0\}$
***disjoint*-Rule**	**if** there occurs v in $\mathcal{L}_E(z_x)$ with $\{R', S'\} \subseteq \alpha(v)$, $R' \sqsubseteq_* R_{xa}$, $S' \sqsubseteq_* R_{xb}$, $x, a, b \in I_\mathcal{A}$, $v \leq 0 \notin \mathcal{L}_E(z_x)$ **then** set $\mathcal{L}_E(z_x) := \mathcal{L}_E(z_x) \cup \{v \leq 0\}$
***equal*-Rule**	**if** there occurs v in $\mathcal{L}_E(z_x)$ with $R' \in \alpha(v)$, $S' \notin \alpha(v)$, $R' \sqsubseteq_* R_{xa}$, $S' \sqsubseteq_* S_{xa}$, $x, a, b \in I_\mathcal{A}$, $v \leq 0 \notin \mathcal{L}_E(z_x)$ **then** set $\mathcal{L}_E(z_x) := \mathcal{L}_E(z_x) \cup \{v \leq 0\}$
***hierarchy*-Rule**	**if** there occurs v in $\mathcal{L}_E(x)$ with $R \in \alpha(v)$, $S \notin \alpha(v)$, $R \sqsubseteq_* S$, $v \leq 0 \notin \mathcal{L}_E(x)$ **then** set $\mathcal{L}_E(x) := \mathcal{L}_E(x) \cup \{v \leq 0\}$
$\geq$**-Rule**	**If** $(\geq nR) \in \mathcal{L}(x)$ and $\xi(R, \geq, n) \notin \mathcal{L}_E(x)$ **then** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{\xi(R, \geq, n)\}$
$\leq$**-Rule**	**If** $(\leq nR) \in \mathcal{L}(x)$ and $\xi(R, \leq, n) \notin \mathcal{L}_E(x)$ **then** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{\xi(R, \leq, n)\}$
***fil*-Rule**	**If** there exists v occurring in $\mathcal{L}_E(z_x)$ such that $\sigma(v) = n$ with $n > 0$: **if** $n = 1$, z_x, z_b root nodes, and $R_{xb} \in \alpha(v)$ with $x, b \in I_\mathcal{A}$ **then if** $\mathcal{L}(\langle z_x, z_b\rangle) = \emptyset$ **then** set $\mathcal{L}(\langle z_x, z_b\rangle) := \alpha(v)$ **end if** **else if** z_x is not blocked and $\neg\exists y : \mathcal{L}(\langle z_x, y\rangle) = \alpha(v)$ **then** create a new node y and set $\mathcal{L}(\langle z_x, y\rangle) := \alpha(v)$ and $card(y) = n$

Fig. 2. Completion rules for $\mathcal{SHQ}$ Abox consistency (listed in decreasing priority)

takes place: (i) The *reset*-Rule becomes applicable for x, which sets $\mathcal{L}_E(x) := \emptyset$ and removes all outgoing edges of x. (ii) Now that $\mathcal{L}_E(x)$ is empty, the $\leq$-Rule and $\geq$-Rule will be invoked again to recompute the partitions and variables. Afterwards, the set of inequations based on the new partitioning is added. (iii) If $(\sigma(v_i) = n) \in \Omega(x)$, where $v_i \in \mathcal{V}_x$ corresponds to the previous partitioning, then set $\mathcal{L}_E(x) := \mathcal{L}_E(x) \cup \{\sum_{v'_j \in \mathcal{V}_{x,i}} v'_j \geq n, \sum_{v'_j \in \mathcal{V}_{x,i}} v'_j \leq n\}$ where $\mathcal{V}_{x,i} := \{v'_j \in \mathcal{V}'_x \mid \alpha(v_i) \subseteq \alpha(v'_j)\}$ and $v'_j \in \mathcal{V}'_x$ are based on the new partitioning. The third task in fact maintains the previous solution in x and records it by means of inequalities in $\mathcal{L}_E(x)$. Therefore, the solution based on the new partitioning will be recreated by the arithmetic reasoner.

An interpretation $\mathcal{I}$ satisfies a $\mathcal{SHQ}$ Abox $\mathcal{A}$ iff the hybrid tableau calculus can derive a complete and clash-free completion forest for $\mathcal{A}$. Due to lack of space we refer the reader to [5] for the proofs on termination, soundness and completeness of the presented calculus.

4 Practical reasoning

4.1 Complexity Analysis

To analyze the complexity of the concept satisfiability test with respect to qualified number restrictions, we count the number of branches that the algorithm creates in the search space. In the following we assume a node $x \in V$ in the completion forest that contains p at-least restrictions and q at-most restrictions in its label ($R_i, R'_j \in N_R$ and $C_i, C'_j \in clos(\mathcal{T})$): $\{\geq n_1\, R_1.C_1, \ldots, \geq n_p\, R_p.C_p\} \subseteq \mathcal{L}(x), \{\leq m_1\, R'_1.C'_1, \ldots, \leq m_q\, R'_q.C'_q\} \subseteq \mathcal{L}(x)$

Standard tableau: The complexity of a standard tableau (e.g., see [9]) can be characterized by $(2^q)^N \times f(N, m)$, where N depends on the biggest number occurring in at-least restrictions and $f(N, m)$ describes the number of possible ways to merge N individuals into m individuals with m the smallest non-zero number occurring in at-most restrictions (see [5] for more details).

Hybrid tableau: During the preprocessing step, the hybrid algorithm converts all qualified number restrictions into unqualified ones which introduces $p + q$ new role names. According to the atomic decomposition presented in Section 3.2, the hybrid algorithm defines $2^{p+q} - 1$ partitions and consequently variables for x; i.e. $|\mathcal{V}_x| = 2^{p+q} - 1$. The ch-Rule opens two branches for each variable in $\mathcal{V}_x$. Therefore, there will be totally $2^{|\mathcal{V}_x|}$ cases to be examined by the arithmetic reasoner. The ch-Rule will be invoked $|\mathcal{V}_x| = 2^{p+q} - 1$ times and creates $2^{2^{p+q}}$ branches in the search space. Hence, the complexity of the algorithm is characterized by a double-exponential function of $p+q$. In [14] a polynomial-time algorithm for integer programming with a fixed number of variables is given. However, our prototype implementation employed the Simplex method which is known to be NP in the worst case but usually behaves very well on average.

The complexity of the standard algorithm is a function of N and therefore the numbers occurring in the at-most restrictions can affect the standard algorithm exponentially. Whereas in the hybrid algorithm, the complexity is independent from N due to its arithmetic approach to the problem.

4.2 Partition Optimization Techniques

Although it seems that the hybrid algorithm is double-exponential in the worst case and the large number of variables seems to be hopelessly inefficient, there are some effective heuristics and optimization techniques which make it feasible to use.

Default Value for Variables: In the semantic branching based on the concept $choose$-Rule (see [9]), in one branch we have C and in the other branch we have $\neg C$ in the label of the nodes. However, due to the ch-Rule (for variables) in one branch we have $v \geq 1$ whereas in the other branch $v \leq 0$. In contrast to concept branching based on the $choose$-Rule, in variable branching we can ignore the existence of variables that are less or equal to zero. In other words, the arithmetic reasoner only considers variables that are greater or equal to one.

Therefore, by setting the default value to $v \leq 0$ for every variable, the algorithm does not need to invoke the ch-Rule $|\mathcal{V}_x|$ times before starting to find a solution for the inequations. More precisely, the algorithm starts with the default value of $v \leq 0$ for all variables in $|\mathcal{V}_x|$. Obviously, the solution for this set of inequations, which is $\forall\, v_i \in \mathcal{V}_x : \sigma(v_i) = 0$, cannot satisfy any at-least restriction. Therefore, the algorithm must choose some variables in $\mathcal{V}_x$ to make them greater or equal to one. Although in the worst case the algorithm still needs to try $2^{|\mathcal{V}_x|}$ cases, by setting this default value it does not need to invoke the ch-Rule when it is not necessary.

We define *don't care* variables as the set of variables that appear in an at-least restriction but in no at-most restriction. These variables have only logical restrictions which later on will be processed by the algorithm. Therefore, any non-negative integer value satisfying all known arithmetic restrictions can be assigned to these variables and we can leave them unchanged in all inequations unless a logical clash is triggered.

We define *satisfying variables* as the set of variables which occur in an at-least restriction and are not *don't care* variables. Since these are the variables that occur in an at-least restriction, by assigning them to be greater or equal to one, the algorithm can lead the arithmetic reasoner to a solution. Whenever a node that is created based on v causes a clash, by means of dependency-directed backtracking we will set $v \leq 0$ and therefore remove v from the *satisfying variables* set. When the *satisfying variables* set becomes empty the algorithm can conclude that the set of qualified number restrictions in $\mathcal{L}(x)$ is unsatisfiable. Notice that the number of variables that can be decided to be greater than zero in an inequation is bounded by the number occurring in their corresponding number restriction.

Variable Encoding: One can encode indices of variables in binary format to easily retrieve the role names related to them. We do not need to assign any memory space for them unless they have a value greater than zero based on an arithmetic solution.

4.3 Dependency-Directed Backtracking

Dependency-directed backtracking or backjumping is a backtracking method which detects the sources of a clash and tries to bypass branching points that are not related to the sources of the clash [12, 11]. In the hybrid algorithm, whenever we encounter a logical clash for a successor y of x, we can conclude that the corresponding variable v_y for the partition in which y resides must be zero. Therefore, we can prune all branches for which $v_y \geq 1 \in \mathcal{L}_E(x)$. This method which is called *simple backtracking* can decrease the size of the search space by pruning half of the branches each time the algorithm detects a clash. For example, for an arbitrary $\mathcal{L}(x)$, by pruning all the branches where $v_y \geq 1$, we will in fact prune $2^{|\mathcal{V}_x|-1} = 2^{2^{p+q}-1}$ branches w.r.t. the *ch*-Rule, which amounts to half of the branches.

We can improve this by a better version in which we prune all branches that have the same reason for the clash caused by v_y. This methods is called *complex backtracking.* For instance, assume the node y that is created based on $\sigma(v_y) = k$ where $k \geq 1$ ends up with a clash. Since we have only one type of clash other than the arithmetic clash, assume the clash is because of $\{A, \neg A\} \subseteq \mathcal{L}(y)$ for some $A \in N_C$. Moreover, assume we know that A is caused by a $\forall R_i.A$ restriction in its predecessor x and $\neg A$ by $\forall S \backslash T_j.(\neg A) \in \mathcal{L}(x)$. It is possible to conclude that all the variables v for which $R_i \in \alpha(v) \wedge T_j \notin \alpha(v)$ will end up with the same clash.

Considering the binary coding for the indices of the variables in which the ith digit represents R_i and the jth digit represents T_j, all the variables, where the ith digit is equal to 1 and the jth digit equal to zero, must be zero. Since the binary coding of the variable indices requires a total of $p+q$ digits, the number of variables that must be zero will be 2^{p+q-2}. The $2^{p+q} - 2^{p+q-2}$ other variables are free to be constrained and will open two branches in the search space. Therefore, the number of branches will be reduced from $2^{|\mathcal{V}_x|}$ to $2^{3/4|\mathcal{V}_x|}$ which is a significant improvement.

5 Evaluation

Our prototype reasoner can process $\mathcal{ALCHQ}$ concept expressions (see also [5]). The system is implemented in Java using the OWL-API 2.1.1 [10]. Qualified number restrictions are expressive constructs added to the forthcoming OWL 2 [17]. Therefore, current OWL benchmarks do not contain qualified number restrictions. In fact, to the best of our knowledge there is no real-world $\mathcal{SHQ}$ benchmark available which contains non-trivial qualified number restrictions. Furthermore, our prototype reasoner does not implement any optimization techniques except the ones introduced above, which target only reasoning with the $\mathcal{Q}$-component of $\mathcal{SHQ}$. Accordingly, we built a set of synthetic test cases to empirically evaluate the hybrid reasoner. We focus our evaluation on concept expressions only containing qualified number restrictions. We identified the following parameters that may affect the complexity of reasoning with number restrictions: (i) The size of numbers occurring in number restrictions. (ii) The number of qualified number restrictions. (iii) The ratio of the number of at-least to the number of at-most restrictions. (iv) Satisfiability versus unsatisfiability of the input concept expression. Due to lack of space we focus in this paper only on the cases (i) and (iv).

Well-known reasoners that support qualified number restrictions such as Racer [7], FaCT++ [19] or Pellet [18] implement numerous optimization techniques. However, to the best of our knowledge, Pellet as well as FaCT++ have no specific optimization techniques for dealing with qualified number restrictions. We decided to select Pellet (version 2.0 RC7 from June 11, 2009) as the representative for a state-of-the-art DL reasoner. We based our evaluations only partially on a comparison with one of the existing reasoners and focused on the study of the behavior of the hybrid reasoner. The experiments were performed under Windows 32 on a standard PC with a dual-core (2.1 GHz) processor and 3 GB of physical memory. Every test was executed in five independent runs (with a timeout of 1000 seconds) and the average of these runs is presented.

In order to assess the scalability of the hybrid algorithm with respect to the size of the numbers, we compared its performance with Pellet. We used the following concept expressions (assuming a role hierarchy $\{R \sqsubseteq T, S \sqsubseteq T, RS \sqsubseteq R, RS \sqsubseteq S\}$; i is a number incremented for each benchmark):

$$(\geq 2i\, RS.(A \sqcup B)) \sqcap (\leq i\, S.A) \sqcap (\leq i\, R.B) \sqcap (\leq (i-1)\, T.(\neg A)) \sqcup (\leq i\, T.(\neg B)) \quad (C_{SAT})$$

$$(\geq 2i\, RS.(A \sqcup B)) \sqcap (\leq i\, S.A) \sqcap (\leq i\, R.B) \sqcap (\leq (i-1)\, T.(\neg A)) \sqcup (\leq (i-1)\, T.(\neg B)) \quad (C_{UNSAT})$$

C_{SAT} is a satisfiable concept while C_{UNSAT} is unsatisfiable. In fact, C_{SAT} is not trivially satisfiable. In the hybrid algorithm, the set of inequations is only satisfied in the case where all variables are zero except $v, v' \geq 1$ and $\alpha(v) = \{RS', S', T'\}$ and $\alpha(v') = \{RS', R'\}$.[1] The standard algorithm in order to examine the satisfiability of

[1] Assume R', S', RS', and T' are new sub-roles of R, S, RS, and T respectively.

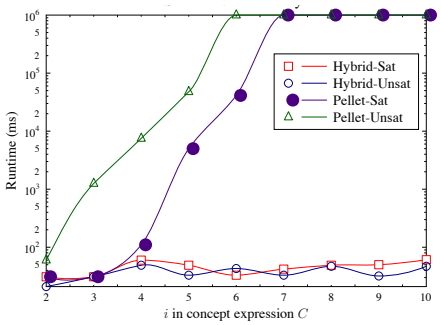


Fig. 3. Increasing the value of numbers

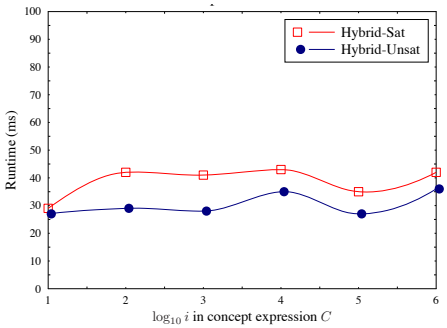


Fig. 4. Exponential growth of i

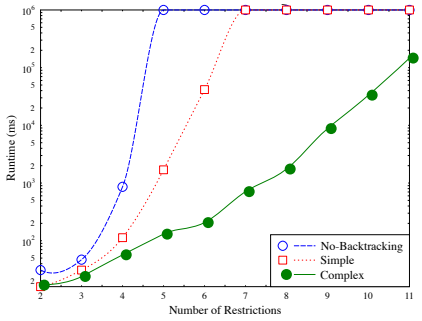


Fig. 5. Different backtracking strategies

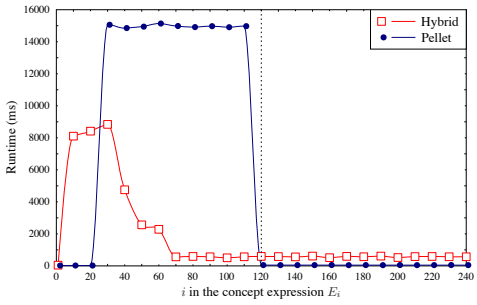


Fig. 6. (Un)Satisfiability of E_i

C_{SAT} creates $(2 \times i)$ RS-successors for x in $(A \sqcup B)$ and according to the three at-most restrictions it opens 8 new branches for each successor. However, since $(2 \times i)$ is much larger than i or $i - 1$, one must start merging the extra successors when an at-most restriction is violated which happens in all branches in this test case. As shown in Figure 3 (logarithmic scale), the growth of i from 2 to 10 has almost no effect on the hybrid reasoner while it dramatically increases the runtime of the standard algorithm for numbers as small as 6 and 7. In contrast to the standard reasoner, the performance of the hybrid reasoner is unaffected by the value of the numbers. Furthermore, to assure that this independence will be preserved also with respect to an exponential growth of i, in Figure 4 we present the performance of the hybrid reasoner for $i = 10^n, n \in 1..6$.

In the next experiment we observed the effect of backtracking on the performance of the hybrid reasoner. In three settings, we first turn off backtracking, secondly enable simple backtracking, and finally utilize complex backtracking (see Section 4.3). We tested an unsatisfiable concept D_{UNSAT} which follows the pattern where $D_j \sqcap D_k = \bot$ for $1 \leq j, k \leq i$, $j \neq k$ with respect to a role hierarchy $R \sqsubseteq T$: $(\geq 3\, R.D_1) \sqcap \ldots \sqcap (\geq 3\, R.D_i) \sqcap (\leq (3i-1)\, T)$. As shown in Figure 5 (logarithmic scale), the high non-determinism of the ch-Rule makes it inevitable to utilize backtracking. Moreover, we can conclude that by using a more comprehensive and informed method of backtracking we can improve the performance significantly.

In the last experiment the test cases are concepts containing four qualified at-least restrictions and one unqualified at-most restriction according to the following pattern. We abbreviate these concepts with E_i where $R \sqsubseteq T$ for $i = 1, 20, 40, \ldots, 220, 240$:

$\geq 30\, R.(B \sqcap C) \sqcap \geq 30\, R.(B \sqcap \neg C) \sqcap \geq 30\, R.(\neg B \sqcap C) \sqcap \geq 30\, R.(\neg B \sqcap \neg C) \sqcap \leq i\, T$. The concept E_i is satisfiable for $i \geq 120$ and unsatisfiable for $i < 120$.

As illustrated in Figure 6, the standard algorithm can easily infer for $i \in 1..29$ that E_i is unsatisfiable since x has at least 30 distinguished successors. However, from E_{30} to E_{120} it becomes very time-consuming for the standard algorithm to merge all 120 successors into i individuals. Moreover, Figure 6 shows that no matter which value i takes between 30 and 119, the standard algorithm performs similarly. In other words, we can conclude that it tries the same number of possible ways of merging which is all the possibilities to merge 4 sets of mutually distinguished individuals. As soon as i becomes greater or equal 120, since the at-most restriction is not violated, the standard algorithm simply ignores it and reasoning becomes trivial for the standard algorithm.

Furthermore, we can conclude from Figure 6 that for the hybrid algorithm $i = 1$ is a trivial case since not more than one variable can have the type of $v \geq 1$ which is the case that easily leads to unsatisfiability for E_1. However, it becomes more difficult as i grows and reaches its maximum for $i = 30$ and starts to decrease gradually until $i = 70$ and remains unchanged until $i = 120$. In fact, this unexpected behavior did not correspond to the formal analysis of the hybrid algorithm. Therefore, we measured the time spent on arithmetic reasoning and logical reasoning as well as the number of different clashes. The reason that $i = 30$ is a breaking point is due to the fact that for $i < 30$ no arithmetic solution exists for the set of inequations. It seems that the increase in runtime is due to this fact and a straight-forward and unoptimized implementation of the arithmetic reasoner. When i grows from 30 to 120, the arithmetic reasoner finds more solutions for the set of inequations which will fail due to logical clashes.

6 Discussion and Future Work

Based on the previous sections we identify the following advantages of the hybrid algorithm: (i) Insensitivity to the value of numbers in number restrictions. (ii) Since all number restrictions are collected before expanding the completion graph, its solution is more probable to survive. (iii) Due to the atomic decomposition the search for a model is semantically structured. When encountering a clash, one can efficiently backtrack to the source of the clash and optimally prune the branches which would lead to the same clash (see Section 4.3).

The following disadvantages of the hybrid algorithm can be observed: (i) According to the nature of the atomic decomposition, the hybrid algorithm introduces an exponential number of variables. (ii) The implemented prototype still requires a preprocessing phase affected by the number of partitions. This initialization time could be saved for trivially satisfiable or unsatisfiable concepts.

We plan to address the following topics in our future work. (i) Since the minimal number of successor property is unnecessary in many cases, one could reduce the arithmetic reasoning time by modifying the goal function and allowing a solution with more successors. (ii) Optimizing the arithmetic reasoner by implementing techniques such as incremental arithmetic reasoning, caching, and heuristics for controlling the order of applying the ch-rule.

Acknowledgements

We express our gratitude to our colleagues Jocelyne Faddoul and Ralf Möller for their valuable advice during this investigation.

References

1. F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
2. Y. Ding. *Tableau-based Reasoning for Description Logics with Inverse Roles and Number Restrictions*. PhD thesis, Department of Computer Science and Software Engineering, Concordia University, April 2008. Available at `http://users.encs.concordia.ca/~haarslev/students/Yu_Ding.pdf`.
3. J. Faddoul, N. Farsinia, V. Haarslev, and R. Möller. A hybrid tableau algorithm for ALCQ. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008), Patras, Greece, July 21-25*, pages 725–726, 2008. An extended version appeared in Proceedings of the 2008 International Workshop on Description Logics (DL-2008), Dresden, Germany, May 13-16, 2008.
4. J. Faddoul, V. Haarslev, and R. Möller. Hybrid reasoning for description logics with nominals and qualified number restrictions. Technical report, Institute for Software Systems (STS), Hamburg University of Technology, 29 pages, 2008. See also http://www.sts.tu-harburg.de/tech-reports/papers.html.
5. N. Farsiniamarj. Combining integer programming and tableau-based reasoning: A hybrid calculus for the description logic $\mathcal{SHQ}$. Master's thesis, Concordia University, Department of Computer Science and Software Engineering, 2008. Available at `http://users.encs.concordia.ca/~haarslev/students/Nasim_Farsinia.pdf`.
6. V. Haarslev and R. Möller. Optimizing reasoning in description logics with qualified number restriction. In *Proceedings of the International Workshop on Description Logics (DL'2001), Aug. 1-3, 2001, Stanford, CA, USA*, pages 142–151, Aug. 2001.
7. V. Haarslev and R. Möller. RACER system description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy*, Lecture Notes in Computer Science, pages 701–705. Springer-Verlag, June 2001.
8. V. Haarslev, M. Timmann, and R. Möller. Combining tableaux and algebraic methods for reasoning with qualified number restrictions. In *Proceedings of the International Workshop on Description Logics (DL'2001), Aug. 1-3, Stanford, USA*, pages 152–161, 2001.
9. B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In J. Allen, R. Fikes, and E. Sandewall, editors, *Second International Conference on Principles of Knowledge Representation, Cambridge, Mass., April 22-25, 1991*, pages 335–346, Apr. 1991.
10. M. Horridge, S. Bechhofer, and O. Noppens. Igniting the OWL 1.1 touch paper: The OWL API. In *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*, volume 258, Innsbruck, Austria, June 2007.
11. I. Horrocks. Backtracking and qualified number restrictions: Some preliminary results. In *Proc. of the 2002 Description Logic Workshop (DL 2002)*, pages 99–106, 2002.
12. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–264, 2000.
13. I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In D. MacAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, Lecture Notes in Computer Science, pages 482–496, Germany, 2000. Springer Verlag.
14. H. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, November 1983.
15. H. Ohlbach and J. Koehler. Role hierarchies and number restrictions. In *Proceedings of the International Workshop on Description Logics (DL-97), Sept. 27 - 29, Gif sur Yvette (Paris), France*, 1997.

16. H. Ohlbach and J. Köhler. Modal logics, description logics and arithmetic reasoning. *Artificial Intelligence*, 109(1-2):1–31, 1999.
17. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax, W3C Working Draft, 02 December 2008, October 2008. `http://www.w3.org/TR/owl2-syntax/` (last visited June 2009).
18. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: a practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2005.
19. D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.