A Hybrid Tableau Algorithm for $\mathcal{ALCQ}$

Jocelyne Faddoul[1] and **Nasim Farsinia**[1] and **Volker Haarslev**[1] and **Ralf Möller**[2]

Abstract. We propose an approach for extending a tableau-based satisfiability algorithm by an arithmetic component. The result is a hybrid concept satisfiability algorithm for the Description Logic (DL) $\mathcal{ALCQ}$ which extends $\mathcal{ALC}$ with *qualified number restrictions*. The hybrid approach ensures a more informed calculus which, on the one hand, adequately handles the interaction between *numerical* and *logical* restrictions of descriptions, and on the other hand, when applied is a very promising framework for average case optimizations.

1 Introduction

Using the DL $\mathcal{ALCQ}$ one can express *numerical restrictions* on concepts with $(\exists R.C)$, $(\geq nR.C)$, and $(\leq mR.C)$ as well as *logical* ones, or both.[3] Such expressiveness means that a satisfiability algorithm for $\mathcal{ALCQ}$ not only needs to satisfy logical restrictions, but also numerical ones.

Our calculus, strongly inspired by [3, 4, 5], consists of a standard tableau for $\mathcal{ALC}$ [1] modified and extended to work with a constraint solver such as a linear inequation solver. The tableau rules encode numerical restrictions into a set of inequations using the *atomic decomposition* technique [5]. The set of inequations is processed by an inequation solver which finds a minimal integer solution (distribution of role fillers) satisfying the numerical restrictions, if one exists. The tableau rules then take care of making sure that such distribution of role fillers also satisfies the logical restrictions. Considering a minimal distribution of fillers ensures that a corresponding model is of minimum size. This hybrid algorithm fills the gaps between tableau algorithms [1] which do not adequately handle numerical reasoning, and the arithmetic reasoning for description logic in [5] where no calculus was proposed and all the input is reduced to equation solving. In contrast with [3] which proposes a recursive algorithm, this hybrid algorithm has the potential to be easily extended to handle more expressive languages.

Since this hybrid algorithm collects all the information about arithmetic expressions before creating any role filler, it will not satisfy an at-least restriction by violating an at-most restriction and there is no need for a mechanism of merging role fillers. Moreover, it reasons about numerical restrictions by means of an inequation solver, thus its performance is not affected by the values of numbers in *qualified number restrictions*. Considering all these features the proposed hybrid algorithm is well suited to improve average case performance.

[1] Concordia University, Montreal, Canada, e-mail: {j_faddou, n_farsin, haarslev}@encs.concordia.ca

[2] Hamburg University of Technology, Hamburg, Germany, e-mail: r.f.moeller@tuhh.de

[3] A universal restriction could be considered as both numerical and logical restriction ($\forall R.C \equiv \leq 0R.(\neg C)$).

2 Arithmetic and Logical Expressions

Given an $\mathcal{ALCQ}$ concept expression with C, D concepts and R a role name, we distinguish between arithmetic and logical expressions. Expressions of the form $(\exists R.C)$, $(\geq nR.C)$, and $(\leq mR.C)$ hold arithmetic restrictions; they specify a lower (upper) bound on the cardinality of the set of *R-fillers*. Expressions of the form $(C \sqcap D)$, $(C \sqcup D)$, and $\neg C$ hold logical restrictions using logical operators on concepts. We refer to these expressions as logical expressions.

In the following, we assume all $\mathcal{ALCQ}$ concept expressions to be in their *negation normal form* (NNF). We use $\dot{\neg}C$ to denote the NNF of $\neg C$. We also define *clos(C)* to be the smallest set of concepts such that: (a) $C \in$ *clos(C)*, (b) if $D \in$ *clos(C)* then $\dot{\neg}D \in$ *clos(C)*, (c) if $(E \sqcap D)$ or $(E \sqcup D) \in$ *clos(C)* then $E, D \in$ *clos(C)*, and (d) if $(\geq nR.E)$ or $(\leq mR.E) \in$ *clos(C)* then $E \in$ *clos(C)*. The size of *clos(C)* is bounded by the size of C. In addition $FIL(R, s)$ is the set of *R-fillers* of an individual $s \in \Delta^{\mathcal{I}}$ for some role $R \in N_{\mathrm{R}}$ and is defined as: *FIL(R,s)* = $\{t \in \Delta^{\mathcal{I}} \mid \langle s, t\rangle \in R^{\mathcal{I}}\}$. The set of all *R-fillers* for R is then defined as *FIL(R)* $= \bigcup_{s \in \Delta^{\mathcal{I}}} FIL(R, s)$.

Re-writing $\mathcal{ALCQ}$ Arithmetic Expressions: We define a concept operator $(\forall(R \backslash S).D)$ and a role implication operator $(R \sqsubseteq S)$ needed to preprocess the input descriptions before applying the calculus. These operators are based on set semantics such that given an interpretation $\mathcal{I}$, then $(\forall(R \backslash S).D)^{\mathcal{I}} = \{s \in \Delta^{\mathcal{I}} \mid \langle s, t\rangle \in R^{\mathcal{I}}$ and $\langle s, t\rangle \notin S^{\mathcal{I}} \Rightarrow t \in D^{\mathcal{I}}\}$ is satisfied and $(R^{\mathcal{I}} \subseteq S^{\mathcal{I}})$ is satisfied for each role implication $R \sqsubseteq S \in \varphi_{\mathrm{r}}$, where φ_{r} is a set of role implications.

Given an $\mathcal{ALCQ}$ concept E and an empty set φ_{r}, we recursively re-write the arithmetic expressions in E such that:

- Each $\geq nR.C$ is replaced with $\geq nR' \sqcap \forall R'.C$, with R' new in N_{R} and $R' \sqsubseteq R$ new in φ_{r}
- Each $\leq mR.D$ is replaced with $\leq mR' \sqcap \forall R'.C \sqcap \forall(R \backslash R').\neg D$, with R' new in N_{R} and $R' \sqsubseteq R$ new in φ_{r}

Satisfiability of Arithmetic Expressions w.r.t φ_{r} Using Linear Inequation Solving: The atomic decomposition technique was used in [4] to reduce reasoning about cardinalities of role fillers to inequation solving. We use the same technique to decide the satisfiability of arithmetic expressions w.r.t φ_{r}. For each role $R \in N_{\mathrm{R}}$ that is involved in an arithmetic expression the introduced sub-role R' enables some hierarchy of roles. We define $H(R) = \{R\} \cup \{R' \mid (R' \sqsubseteq R) \in \varphi_{\mathrm{r}}\}$ as the role hierarchy of R.

For every role $R' \in H(R)$, the set of *R'-fillers* forms a subset of the set of *R-fillers* ($FIL(R') \subseteq FIL(R)$). Using the atomic decomposition of $H(R)$ we can define all possible intersections between *R-fillers* as disjoint partitions. Each $L \subseteq H(R)$ is associated with a unique partition $P(L) = \bigcap_{\mathrm{R}' \in \mathrm{L}} FIL(R')$. Furthermore, $\mathcal{P}$ is the set of partitions defined for the decompositions of all hierarchies in φ_{r}:

$$\mathcal{P} = \bigcup_{\mathrm{R}\in\mathrm{N_R}} \left(\begin{array}{c} \{L \mid L \subseteq H(R)\} \setminus \\ \{L \mid L \subseteq H(R), \exists R' \in L, R \notin L \text{ and} \\ R' \sqsubseteq R \in \varphi_{\mathrm{r}}\} \end{array} \right)$$

We do not consider L such that $R' \in L$, $R \notin L$ for some $(R' \sqsubseteq R) \in \varphi_r$ since the corresponding partition $P(L)$ does not satisfy $FIL(R') \subseteq FIL(R)$ and therefore must be empty.

We assign a variable name v for each partition $P(L)$ such that v is mapped to a non-negative integer value n which denotes the cardinality of $P(L)$. Let $\mathcal{V}$ be the set of all variable names, we maintain a mapping between variable names and their corresponding partitions using a function α: $\mathcal{V} \rightarrow \mathcal{P}$ such that for some non-negative integer n assigned to a variable v we have $n = \#P(\alpha(v))$.

Since the partitions are mutually disjoint and the cardinality function is additive, a lower (upper) bound n (m) on the cardinality of the set of role fillers $FIL(S)$ for some role $S \in H(R)$ can be reduced to an inequation of the form $\sum_{v\in V_S} v \geq n$ $(\sum_{v\in V_S} v \leq m)$. V_S denotes the set of variable names mapped to partitions for a role S and is defined as $V_S = \{v \in \mathcal{V} \mid S \in \alpha(v)\}$. Thus, we can easily convert an expression of the form $(\geq nS)$ or $(\leq mS)$ into an inequation using ξ such that $\xi(S, \geq, n) = \sum_{v\in V_S} v \geq n$, and $\xi(S, \leq, m) = \sum_{v\in V_S} v \leq m$. Each variable v occurring in an inequation can be mapped to a non-negative integer p such that assuming $\alpha(v) = \{R', R''\}$, this means that $\#(FIL(R') \cap FIL(R'')) = p$ and the corresponding partition $P(\alpha(v))$ must have p fillers.

3 A Hybrid Tableau Algorithm for $\mathcal{ALCQ}$

In general, logical and arithmetic expressions in φ share symbols, therefore, their satisfiability cannot be decided independently. Furthermore, disjunctions in φ need to be treated case by case. For this purpose we propose a tableau-based hybrid algorithm which decides the existence of a tableau for an $\mathcal{ALCQ}$ concept expression φ.

A *completion graph* is a directed graph $G = (V, E, \mathcal{L}, \mathcal{L}_E)$ where each node $x \in V$ is labeled with $\mathcal{L}$ and $\mathcal{L}_E$ such that $\mathcal{L}(x)$ denotes a set of concept expressions, $\mathcal{L}(x) \subseteq clos(\varphi)$, and $\mathcal{L}_E(x)$ denotes a set of inequations. Each edge $\langle x, y\rangle \in E$ is labeled with a set, $\mathcal{L}(\langle x, y\rangle) \subseteq \mathcal{P}$, of role names.

We denote by ξ_x the set of inequations in $\mathcal{L}_E(x)$ obtained by converting the at-least and at-most restrictions in $\mathcal{L}(x)$. An integer solution σ for ξ_x maps each variable v occurring in ξ_x to a non-negative integer p such that σ is a distribution of role fillers of x. The distribution is consistent with the lower and upper bounds expressed in arithmetic restrictions and the hierarchy in φ_{r}.

A node x in V is said to contain a clash if either (i) $\{C, \doteq C\} \subseteq L(x)$, or (ii) the set of inequations $\xi_x \subseteq \mathcal{L}_E(x)$ does not admit a non-negative integer solution. Case (ii) is decided by the inequation solver. When no rules are applicable or there is a clash, a *completion graph* is said to be *complete*.

To decide the satisfiability of a concept expression φ, the algorithm starts with the completion graph $G = (\{x\}, \emptyset, \{\varphi\}, \emptyset)$. G is then expanded by applying the expansion rules given in Fig. 1 until no more rules are applicable or a clash occurs. When G is complete and there is no clash, this means that the arithmetic expressions are satisfied as well as the logical ones: we have a pre-model and the algorithm returns that φ is satisfiable.

Explaining the Rules: We assign the *fil-Rule* the lowest priority; All other rules can be fired in arbitrary order. The $\leq$-*Rule* and the $\geq$-*Rule* are responsible for encoding the arithmetic expressions in the label $\mathcal{L}(x)$ of a node x into a set (ξ_x) of inequations maintained in $\mathcal{L}_{\mathrm{E}}(x)$. An inequation solver is always active and is responsible for finding a non-negative integer solution σ for ξ_x or triggering a clash if no solution is possible.

$\sqcap$**-Rule**	**If** $C \sqcap D \in \mathcal{L}(x)$, and $\{C, D\} \nsubseteq \mathcal{L}(x)$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C, D\}$
$\sqcup$**-Rule**	**If** $C \sqcup D \in \mathcal{L}(x)$, and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ **or** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{D\}$
$\forall$**-Rule**	**If** $\forall R.C \in \mathcal{L}(x)$ and $R \in \mathcal{L}(\langle x, y\rangle)$ with $C \notin \mathcal{L}(y)$ **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 y such that $R \in \mathcal{L}(\langle x, y\rangle)$ and $S \notin \mathcal{L}(\langle x, y\rangle)$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\leq$**-Rule**	**If** $(\leq nR) \in \mathcal{L}(x)$ and $\xi(R, \leq, n) \notin \mathcal{L}_E(x)$ **then** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{\xi(R, \leq, n)\}$
***ch*-Rule**	**If** there exists v occurring in $\mathcal{L}_E(x)$ with $\{v \geq 1, v \leq 0\} \cap \mathcal{L}_E(x) = \emptyset$ **then** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{v \geq 1\}$ **or** 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 $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{\xi(R, \geq, n)\}$
***fil*-Rule**	**If** there exists v occurring in $\mathcal{L}_E(x)$ such that (i) $\sigma(v) = m$ with $m > 0$, and (ii) there are no m nodes $y_1 \ldots y_m$ with $\mathcal{L}(\langle x, y_i\rangle) = \alpha(v)$ for $1 \leq i \leq m$ **then** create m new nodes $y_1 \ldots y_m$ and set $\mathcal{L}(\langle x, y_i\rangle) = \alpha(v)$ for $1 \leq i \leq m$

Figure 1. Expansion rules for $\mathcal{ALCQ}$.

The *ch-Rule* is used to check for empty partitions. Given a set of inequations in the label $(\mathcal{L}_{\mathrm{E}})$ of a node x and a variable v corresponding to a partition $\alpha(v)$ in $\mathcal{P}$, we distinguish between two cases: (i) The case when a partition must be empty; this can happen when restrictions of individuals assigned to this partition trigger a clash. (ii) The case when a partition can have at least one individual; if a partition can have one individual without causing any logical clash, this means that we can have m $(m \geq 1)$[4] individuals also in this partition without a clash. Since the inequation solver is unaware of logical restrictions of filler domains we allow an explicit distinction between cases (i) and (ii). We do this by non-deterministically assigning ≤ 0 or ≥ 1 for each variable v occurring in $\mathcal{L}_{\mathrm{E}}(x)$. The *fil-Rule* is used to generate role fillers for a node x depending on the distribution (solution) returned by the inequation solver. For a proof of correctness we refer the reader to [2].

REFERENCES

[1] F. Baader and U. Sattler, 'An overview of tableau algorithms for description logics', *Studia Logica*, **69**, 5–40, (2001).

[2] J. Faddoul, N. Farsinia, V. Haarslev, and R. Möller, 'A Hybrid Tableau Algorithm for $\mathcal{ALCQ}$', in *Description Logics*, (2008).

[3] V. Haarslev, M. Timmann, and R. Möller, 'Combining tableaux and algebraic methods for reasoning with qualified number restrictions', in *Description Logics*, pp. 152–161, (2001).

[4] H.J. Ohlbach and J. Koehler, 'Role hierarchies and number restrictions', in *Description Logics*, (1997).

[5] H.J. Ohlbach and J. Koehler, 'Modal logics description logics and arithmetic reasoning', *Artificial Intelligence*, **109**(1-2), 1–31, (1999).

[4] The value of m is decided by the inequation solver.