`DysToPic`: a Multi-Engine Theorem Prover for Preferential Description Logics

Laura Giordano[1], Valentina Gliozzi[2],
Nicola Olivetti[3], Gian Luca Pozzato[2], and Luca Violanti[4]

[1] DISIT - U. Piemonte Orientale - Alessandria, Italy - `laura.giordano@unipmn.it`
[2] Dip. Informatica - Universitá di Torino - Italy
`{valentina.gliozzi,gianluca.pozzato}@unito.it`
[3] Aix Marseille Université - ENSAM, Université de Toulon, LSIS UMR 7296 - France
`nicola.olivetti@univ-amu.fr`
[4] NCR Edinburgh - United Kingdom - `luca.violanti@gmail.com`

Abstract. We describe `DysToPic`, a theorem prover for the preferential Description Logic $\mathcal{ALC} + \mathbf{T}_{min}$. This is a nonmonotonic extension of standard $\mathcal{ALC}$ based on a typicality operator $\mathbf{T}$, which enjoys a preferential semantics. `DysToPic` is a multi-engine Prolog implementation of a labelled, two-phase tableaux calculus for $\mathcal{ALC} + \mathbf{T}_{min}$ whose basic idea is that of performing these two phases by different machines. The performances of `DysToPic` are promising, and significantly better than the ones of its predecessor `PreDeLo 1.0`.

1 Introduction

Nonmonotonic extensions of Description Logics (DLs) have been actively investigated since the early 90s [3, 1, 4, 6, 7, 13, 5]. A simple but powerful nonmonotonic extension of DLs is proposed in [7, 13, 12]: in this approach "typical" or "normal" properties can be directly specified by means of a "typicality" operator $\mathbf{T}$ enriching the underlying DL; the typicality operator $\mathbf{T}$ is essentially characterized by the core properties of nonmonotonic reasoning axiomatized by either *preferential logic* [14] or *rational logic* [15]. In these logics one can consistently express defeasible inclusions and exceptions such as "normally, kids eat chocolate, but typical kids who have lactose intolerance do not":

$\mathbf{T}(\mathit{Kid}) \sqsubseteq \mathit{ChocolateEater}$
$\mathbf{T}(\mathit{Kid} \sqcap \exists \mathit{HasIntolerance}.\mathit{Lactose}) \sqsubseteq \neg \mathit{ChocolateEater}$.

In order to perform useful inferences, in [13] we have introduced a nonmonotonic extension of the logic $\mathcal{ALC}$ plus $\mathbf{T}$ based on a minimal model semantics. Intuitively, the idea is to restrict our consideration to models that maximize typical instances of a concept: more in detail, we introduce a preference relation among $\mathcal{ALC}$ plus $\mathbf{T}$ models, then we define a *minimal entailment* restricted to models that are minimal with respect to such preference relation. The resulting logic, called $\mathcal{ALC} + \mathbf{T}_{min}$, supports typicality assumptions, so that if one knows that Roman is a kid, one can nonmonotonically assume that he is also a *typical* kid and therefore that he eats chocolate. As an example, for a TBox specified by the inclusions above, in $\mathcal{ALC} + \mathbf{T}_{min}$ we can infer that:

TBox $\models_{\mathcal{ALC}+\mathbf{T}_{min}} \mathbf{T}(\mathit{Kid} \sqcap \mathit{Tall}) \sqsubseteq \mathit{ChocolateEater}$[5]
TBox $\cup \{\mathit{Kid}(\mathit{daniel})\} \models_{\mathcal{ALC}+\mathbf{T}_{min}} \mathit{ChocolateEater}(\mathit{daniel})$
TBox $\cup \{\mathit{Kid}(\mathit{daniel}), \exists \mathit{HasIntolerance}.\mathit{Lactose}(\mathit{daniel})\} \models_{\mathcal{ALC}+\mathbf{T}_{min}}$

[5] Being tall is irrelevant with respect to eating chocolate or not.

$\models_{\mathcal{ALC}+\mathbf{T}_{min}} \neg \mathit{ChocolateEater}(\mathit{daniel})$[6]
TBox $\cup$ $\{\mathit{Kid}(\mathit{daniel}), \mathit{Tall}(\mathit{daniel})\} \models_{\mathcal{ALC}+\mathbf{T}_{min}} \mathit{ChocolateEater}(\mathit{daniel})$
TBox $\cup \{\exists \mathit{HasBrother}.\ \mathit{Kid}(\mathit{seth})\} \models_{\mathcal{ALC}+\mathbf{T}_{min}} \exists\ \mathit{HasBrother}.\ \mathit{ChocolateEater}(\mathit{seth})$[7]

In this work we focus on theorem proving for nonmonotonic extensions of DLs. We introduce `DysToPic`, a theorem prover for preferential Description Logic $\mathcal{ALC}+\mathbf{T}_{min}$. `DysToPic` implements the labelled tableaux calculus for this logic introduced in [13] performing a two-phase computation: in the first phase, candidate models falsifying a given query are generated (complete open branches); in the second phase the minimality of candidate models is checked by means of an auxiliary tableau construction. `DysToPic` is a multi-engine theorem prover, whose basic idea is that the two phases of the calculus are performed by different machines: a "master" machine M, called the *employer*, executes the first phase of the tableaux calculus, whereas other computers are used to perform the second phase on open branches detected by M. When M finds an open branch, it invokes the second phase on the calculus on a different "slave" machine, called *worker*, S_1, while M goes on performing the first phase on other branches, rather than waiting for the result of S_1. When another open branch is detected, then another machine S_2 is involved in the procedure in order to perform the second phase of the calculus on that branch. In this way, the second phase is performed simultaneously on different branches, leading to a significant increase of the performances.

Labelled tableaux calculi are implemented in Prolog, following the line of the predecessor `PreDeLo 1.0`, introduced in [11]: `DysToPic` is inspired by the methodology introduced by the system leanT^AP [2], even if it does not fit its style in a rigorous manner. The basic idea is that each axiom or rule of the tableaux calculi is implemented by a Prolog clause of the program: the resulting code is therefore simple and compact.

In general, the literature contains very few proof methods for nonmonotonic extensions of DLs. We provide some experimental results to show that the performances of `DysToPic` are promising, in particular comparing them to the ones of `PreDeLo 1.0`. `DysToPic` is available for free download at:

`http://www.di.unito.it/~pozzato/theoremprovers.html`

2 The Logic $\mathcal{ALC}+\mathbf{T}_{min}$

The logic $\mathcal{ALC}+\mathbf{T}_{min}$ is obtained by adding to the standard $\mathcal{ALC}$ the typicality operator $\mathbf{T}$ [7, 12]. The intuitive idea is that $\mathbf{T}(C)$ selects the *typical* instances of a concept C. We can therefore distinguish between the properties that hold for all instances of concept C ($C \sqsubseteq D$), and those that only hold for the normal or typical instances of C ($\mathbf{T}(C) \sqsubseteq D$).

The language $\mathcal{L}$ of the logic is defined by distinguishing *concepts* and *extended concepts* as follows. Given an alphabet of concept names $\mathcal{C}$, of role names $\mathcal{R}$, and of individual constants $\mathcal{O}$, $A \in \mathcal{C}$ and $\top$ are *concepts* of $\mathcal{L}$; if $C, D \in \mathcal{L}$ and $R \in \mathcal{R}$, then $C \sqcap D, C \sqcup D, \neg C, \forall R.C, \exists R.C$ are *concepts* of $\mathcal{L}$. If C is a concept, then C and $\mathbf{T}(C)$ are *extended concepts*, and all the boolean combinations of extended concepts

[6] Giving preference to more specific information.
[7] Minimal consequence applies to individuals not explicitly named in the ABox as well, without any ad-hoc mechanism.

are extended concepts of $\mathcal{L}$. A KB is a pair (TBox,ABox). TBox contains inclusion relations (subsumptions) $C \sqsubseteq D$, where C is an extended concept of the form either C' or $\mathbf{T}(C')$, and $D \in \mathcal{L}$ is a concept. ABox contains expressions of the form $C(a)$ and $R(a, b)$, where $C \in \mathcal{L}$ is an extended concept, $R \in \mathcal{R}$, and $a, b \in \mathcal{O}$.

In order to provide a semantics to the operator $\mathbf{T}$, we extend the definition of a model used in "standard" logic $\mathcal{ALC}$. The idea is that the operator $\mathbf{T}$ is characterized by a set of postulates that are essentially a reformulation of the Kraus, Lehmann and Magidor's axioms of *preferential logic* **P** [14]. Intuitively, the assertion $\mathbf{T}(C) \sqsubseteq D$ corresponds to the conditional assertion $C \mid\!\sim D$ of **P**. $\mathbf{T}$ has therefore all the "core" properties of nonmonotonic reasoning as it is axiomatized by **P**. The idea is that there is a global preference relation among individuals, in the sense that $x < y$ means that x is "more normal" than y, and that the typical members of a concept C are the minimal elements of C with respect to this relation. In this framework, an element $x \in \Delta$ is a *typical instance* of some concept C if $x \in C^I$ and there is no element in C^I *more typical* than x. The typicality preference relation is partial.

Definition 1. *Given an irreflexive and transitive relation* $<$ *over* Δ*, we define* $Min_<(S) = \{x : x \in S$ *and* $\nexists y \in S$ *s.t.* $y < x\}$. *We say that* $<$ *is well-founded if and only if, for all* $S \subseteq \Delta$*, for all* $x \in S$*, either* $x \in Min_<(S)$ *or* $\exists y \in Min_<(S)$ *such that* $y < x$.

Definition 2. *A model of* $\mathcal{ALC} + \mathbf{T}_{min}$ *is any structure* $\langle \Delta, <, I\rangle$*, where:* Δ *is the domain;* I *is the extension function that maps each extended concept* C *to* $C^I \subseteq \Delta$*, and each role* R *to a* $R^I \subseteq \Delta \times \Delta$*;* $<$ *is an irreflexive, transitive and well-founded (Definition 1) relation over* Δ*.* I *is defined in the usual way (as for* $\mathcal{ALC}$*) and, in addition,* $(\mathbf{T}(C))^I = Min_<(C^I)$.

Given a model $\mathcal{M}$ of Definition 2, I can be extended so that it assigns to each individual a of $\mathcal{O}$ a distinct element a^I of the domain Δ (unique name assumption). We say that $\mathcal{M}$ satisfies an inclusion $C \sqsubseteq D$ if $C^I \subseteq D^I$, and that $\mathcal{M}$ satisfies $C(a)$ if $a^I \in C^I$ and $R(a, b)$ if $(a^I, b^I) \in R^I$. Moreover, $\mathcal{M}$ satisfies TBox if it satisfies all its inclusions, and $\mathcal{M}$ satisfies ABox if it satisfies all its formulas. $\mathcal{M}$ satisfies a KB (TBox,ABox), if it satisfies both TBox and ABox.

The semantics of the typicality operator can be specified by modal logic. The interpretation of $\mathbf{T}$ can be split into two parts: for any x of the domain Δ, $x \in (\mathbf{T}(C))^I$ just in case (i) $x \in C^I$, and (ii) there is no $y \in C^I$ such that $y < x$. Condition (ii) can be represented by means of an additional modality $\Box$, whose semantics is given by the preference relation $<$ interpreted as an accessibility relation. The interpretation of $\Box$ in $\mathcal{M}$ is as follows: $(\Box C)^I = \{x \in \Delta \mid$ for every $y \in \Delta$, if $y < x$ then $y \in C^I\}$. We immediately get that $x \in (\mathbf{T}(C))^I$ if and only if $x \in (C \sqcap \Box\neg C)^I$.

Even if the typicality operator $\mathbf{T}$ itself is nonmonotonic (i.e. $\mathbf{T}(C) \sqsubseteq E$ does not imply $\mathbf{T}(C \sqcap D) \sqsubseteq E$), what is inferred from a KB can still be inferred from any KB' with KB $\subseteq$ KB'. In order to perform *nonmonotonic* inferences, in [13] we have strengthened the above semantics by restricting entailment to a class of minimal (or preferred) models. Intuitively, the idea is to restrict our consideration to models that *minimize the non-typical instances of a concept*.

Given a KB, we consider a finite set $\mathcal{L}_{\mathbf{T}}$ of concepts: these are the concepts whose non-typical instances we want to minimize. We assume that the set $\mathcal{L}_{\mathbf{T}}$ contains at least all concepts C such that $\mathbf{T}(C)$ occurs in the KB or in the query F, where a *query* F is either an assertion $C(a)$ or an inclusion relation $C \sqsubseteq D$. As we have just said, $x \in C^I$ is typical for C if $x \in (\Box\neg C)^I$. Minimizing the non-typical instances of C therefore means to minimize the objects falsifying $\Box\neg C$ for $C \in \mathcal{L}_{\mathbf{T}}$. Hence, for a model $\mathcal{M} = \langle \Delta, <, I\rangle$, we define $\mathcal{M}^{\Box^-}_{\mathcal{L}_{\mathbf{T}}} = \{(x, \neg\Box\neg C) \mid x \not\in (\Box\neg C)^I$, with $x \in \Delta, C \in \mathcal{L}_{\mathbf{T}}\}$.

Definition 3 (Preferred and minimal models). *Given a model $\mathcal{M} = \langle \Delta, <, I\rangle$ of a knowledge base* KB*, and a model $\mathcal{M}' = \langle \Delta', <', I'\rangle$ of* KB*, we say that $\mathcal{M}$ is preferred to $\mathcal{M}'$ w.r.t. $\mathcal{L}_{\mathbf{T}}$, and we write $\mathcal{M} <_{\mathcal{L}_{\mathbf{T}}} \mathcal{M}'$, if (i) $\Delta = \Delta'$, (ii) $\mathcal{M}^{\Box^-}_{\mathcal{L}_{\mathbf{T}}} \subset \mathcal{M}'^{\Box^-}_{\mathcal{L}_{\mathbf{T}}}$, (iii) $a^I = a^{I'}$ for all $a \in \mathcal{O}$. $\mathcal{M}$ is a* minimal model *for* KB *(w.r.t. $\mathcal{L}_{\mathbf{T}}$) if it is a model of* KB *and there is no other model $\mathcal{M}'$ of* KB *such that $\mathcal{M}' <_{\mathcal{L}_{\mathbf{T}}} \mathcal{M}$.*

Definition 4 (Minimal Entailment in $\mathcal{ALC} + \mathbf{T}_{min}$). *A query F is minimally entailed in $\mathcal{ALC} + \mathbf{T}_{min}$ by* KB *with respect to $\mathcal{L}_{\mathbf{T}}$ if F is satisfied in all models of* KB *that are minimal with respect to $\mathcal{L}_{\mathbf{T}}$. We write* KB $\models_{\mathcal{ALC}+\mathbf{T}_{min}} F$.

As an example, consider the TBox of the Introduction.
We have that TBox $\cup$ $\{Kid(daniel)\}$ $\models_{\mathcal{ALC}+\mathbf{T}_{min}}$ $ChocolateEater(daniel)$, since $daniel^I \in (Kid \sqcap \Box\neg Kid)^I$ for all minimal models $\mathcal{M} = \langle \Delta <, I\rangle$ of the TBox. In contrast, by the nonmonotonic character of minimal entailment, TBox $\cup$ $\{Kid(daniel)$, $\exists HasIntolerance.Lactose(daniel)\}$ $\models_{\mathcal{ALC}+\mathbf{T}_{min}}$ $\neg ChocolateEater(daniel)$.

3 A Tableau Calculus for $\mathcal{ALC} + \mathbf{T}_{min}$

In this section we recall the tableau calculus $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{min}$ for deciding whether a query F is minimally entailed from a KB in $\mathcal{ALC} + \mathbf{T}_{min}$ introduced in [13]. The calculus performs a two-phase computation: in the first phase, a tableau calculus, called $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH1}$, simply verifies whether KB $\cup$ $\{\neg F\}$ is satisfiable in a model of Definition 2, building candidate models; in the second phase another tableau calculus, called $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH2}$, checks whether the candidate models found in the first phase are *minimal* models of KB, i.e. for each open branch of the first phase, $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH2}$ tries to build a model of KB which is preferred to the candidate model w.r.t. Definition 3. The whole procedure is formally defined at the end of this section (Definition 5).

$\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{min}$ tries to build an open branch representing a minimal model satisfying KB $\cup$ $\{\neg F\}$, where $\neg F$ is the negation of the query F and is defined as follows: if $F \equiv C(a)$, then $\neg F \equiv (\neg C)(a)$; if $F \equiv C \sqsubseteq D$, then $\neg F \equiv (C \sqcap \neg D)(x)$, where x does not occur in KB. $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{min}$ makes use of labels, denoted with $x, y, z, \ldots$. Labels represent individuals either named in the ABox or implicitly expressed by existential restrictions. These labels occur in *constraints*, that can have the form $x \xrightarrow{R} y$ or $x : C$, where x, y are labels, R is a role and C is a concept of $\mathcal{ALC} + \mathbf{T}_{min}$ or has the form $\Box\neg D$ or $\neg\Box\neg D$, where D is a concept.

3.1 The Tableaux Calculus $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH1}$

A tableau of $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH1}$ is a tree whose nodes are pairs $\langle S \mid U\rangle$. S is a set of constraints, whereas U contains formulas of the form $C \sqsubseteq D^L$, representing inclusion relations $C \sqsubseteq D$ of the TBox. L is a list of labels, used in order to ensure the termination of the tableau calculus. A branch is a sequence of nodes $\langle S_1 \mid U_1\rangle, \langle S_2 \mid$

$U_2\rangle, \ldots, \langle S_n \mid U_n\rangle \ldots$, where each node $\langle S_i \mid U_i\rangle$ is obtained from its immediate predecessor $\langle S_{i-1} \mid U_{i-1}\rangle$ by applying a rule of $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$, having $\langle S_{i-1} \mid U_{i-1}\rangle$ as the premise and $\langle S_i \mid U_i\rangle$ as one of its conclusions. A branch is closed if one of its nodes is an instance of a (Clash) axiom, otherwise it is open. A tableau is closed if all its branches are closed.

The rules of $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ are presented in Fig. 1. Rules $(\exists^+)$ and $(\Box^-)$ are called *dynamic* since they can introduce a new variable in their conclusions. The other rules are called *static*. We do not need any extra rule for the positive occurrences of $\Box$, since these are taken into account by the computation of $S^M_{x\to y}$ of $(\Box^-)$. The (cut) rule ensures that, given any concept $C \in \mathcal{L}_\mathbf{T}$, an open branch built by $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ contains either $x : \Box\neg C$ or $x : \neg\Box\neg C$ for each label x: this is needed in order to allow $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ to check the minimality of the model corresponding to the open branch. As mentioned above, given a node $\langle S \mid U\rangle$, each formula $C \sqsubseteq D$ in U is equipped with the list L of labels to which the rule $(\sqsubseteq)$ has already been applied. This avoids multiple applications of such rule to the same subsumption by using the same label.

In order to check the satisfiability of a KB, we build its *corresponding constraint system* $\langle S \mid U\rangle$, and we check its satisfiability. Given KB=(TBox,ABox), its *corresponding constraint system* $\langle S \mid U\rangle$ is defined as follows: $S = \{a : C \mid C(a) \in \text{ABox}\} \cup \{a \xrightarrow{R} b \mid R(a,b) \in \text{ABox}\}$; $U = \{C \sqsubseteq D^\emptyset \mid C \sqsubseteq D \in \text{TBox}\}$. KB is satisfiable if and only if its corresponding constraint system $\langle S \mid U\rangle$ is satisfiable. In order to verify the satisfiability of KB $\cup \{\neg F\}$, we use $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ to check the satisfiability of the constraint system $\langle S \mid U\rangle$ obtained by adding the constraint corresponding to $\neg F$ to S', where $\langle S' \mid U\rangle$ is the corresponding constraint system of KB. To this purpose, the rules of the calculus $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ are applied until either a contradiction is generated (*clash*) or a model satisfying $\langle S \mid U\rangle$ can be obtained.

The rules of $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ are applied with the following *standard strategy*: 1. apply a rule to a label x only if no rule is applicable to a label y such that $y \prec x$ (where $y \prec x$ says that label x has been introduced in the tableaux later than y); 2. apply dynamic rules only if no static rule is applicable.

Theorem 1. *Given $\mathcal{L}_\mathbf{T}$, KB $\models_{\mathcal{ALC}+\mathbf{T}_{min}} F$ if and only if there is no open branch* **B** *in the tableau built by $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ for the constraint system corresponding to KB $\cup \{\neg F\}$ such that the model represented by* **B** *is a minimal model of KB.*

Thanks to the side conditions on the application of the rules and the blocking machinery adopted by the dynamic ones, in [13] it has been shown that any tableau generated by $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ for $\langle S \mid U\rangle$ is finite.

3.2 The Tableaux Calculus $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$

Let us now introduce the calculus $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ which checks whether each open branch **B** built by $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ represents a minimal model of the KB.

Given an open branch **B** of a tableau built from $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$, let $\mathcal{D}(\mathbf{B})$ be the set of labels occurring in **B**. Moreover, let $\mathbf{B}^{\Box^-}$ be the set of formulas $x : \neg\Box\neg C$ occurring in **B**, that is to say $\mathbf{B}^{\Box^-} = \{x : \neg\Box\neg C \mid x : \neg\Box\neg C \text{ occurs in } \mathbf{B}\}$.

$$\langle S, x : C, x : \neg C \mid U\rangle\ \text{(Clash)} \qquad \langle S, x : \neg\top \mid U\rangle\ \text{(Clash)}_{\top} \qquad \langle S, x : \bot \mid U\rangle\ \text{(Clash)}_{\bot}$$

$$\frac{\langle S, x : \neg(C \sqcap D) \mid U\rangle}{\langle S, x : \neg(C \sqcap D), x : \neg C \mid U\rangle \qquad \langle S, x : \neg(C \sqcap D), x : \neg D \mid U\rangle}\ (\sqcap^-) \quad \text{if } x : \neg C \notin S \text{ and } x : \neg D \notin S$$

$$\frac{\langle S, x : C \sqcap D \mid U\rangle}{\langle S, x : C \sqcap D, x : C, x : D \mid U\rangle}\ (\sqcap^+) \quad \text{if } \{x : C, x : D\} \not\subseteq S$$

$$\frac{\langle S, x : C \sqcup D \mid U\rangle}{\langle S, x : C \sqcup D, x : C \mid U\rangle \qquad \langle S, x : C \sqcup D, x : D \mid U\rangle}\ (\sqcup^+) \quad \text{if } x : C \notin S \text{ and } x : D \notin S$$

$$\frac{\langle S, x : \neg(C \sqcup D) \mid U\rangle}{\langle S, x : \neg(C \sqcup D), x : \neg C, x : \neg D \mid U\rangle}\ (\sqcup^-) \quad \text{if } \{x : \neg C, x : \neg D\} \not\subseteq S$$

$$\frac{\langle S, x : \neg\neg C \mid U\rangle}{\langle S, x : \neg\neg C, x : C \mid U\rangle}\ (\neg) \quad \text{if } x : C \notin S$$

$$\frac{\langle S, x : \mathbf{T}(C) \mid U\rangle}{\langle S, x : \mathbf{T}(C), x : C, x : \Box\neg C \mid U\rangle}\ (\mathbf{T}^+) \quad \text{if } \{x : C, x : \Box\neg C\} \not\subseteq S$$

$$\frac{\langle S, x : \neg\mathbf{T}(C) \mid U\rangle}{\langle S, x : \neg\mathbf{T}(C), x : \neg C \mid U\rangle \qquad \langle S, x : \neg\mathbf{T}(C), x : \neg\Box\neg C \mid U\rangle}\ (\mathbf{T}^-) \quad \text{if } x : \neg C \notin S \text{ and } x : \neg\Box\neg C \notin S$$

$$\frac{\langle S \mid U\rangle}{\langle S, x : \Box\neg C \mid U\rangle \qquad \langle S, x : \neg\Box\neg C \mid U\rangle}\ (cut) \quad \begin{array}{r}\text{if } x : \neg\Box\neg C \notin S \text{ and } x : \Box\neg C \notin S\\ C \in \mathcal{L}_{\mathbf{T}}\\ x \text{ occurs in } S\end{array}$$

$$\frac{\langle S \mid U, C \sqsubseteq D^L\rangle}{\langle S, x : \neg C \sqcup D \mid U, C \sqsubseteq D^{L,x}\rangle}\ (\sqsubseteq) \quad \text{if } x \text{ occurs in } S \text{ and } x \notin L$$

$$\frac{\langle S, x : \forall R.C, x \xrightarrow{R} y \mid U\rangle}{\langle S, x : \forall R.C, x \xrightarrow{R} y, y : C \mid U\rangle}\ (\forall^+) \quad \text{if } y : C \notin S$$

$$\frac{\langle S, x : \exists R.C \mid U\rangle}{\langle S, x : \exists R.C, x \xrightarrow{R} y, y : C \mid U\rangle \quad \langle S, x : \exists R.C, x \xrightarrow{R} v_1, v_1 : C \mid U\rangle \quad \langle S, x : \exists R.C, x \xrightarrow{R} v_2, v_2 : C \mid U\rangle \cdots \langle S, x : \exists R.C, x \xrightarrow{R} v_n, v_n : C \mid U\rangle}\ (\exists^+)$$
$$\begin{array}{r}y \text{ new}\\ \text{if } \nexists z \prec x \text{ s.t. } z \equiv_{S,x:\exists R.C} x \text{ and } \nexists u \text{ s.t. } x \xrightarrow{R} u \in S \text{ and } u : C \in S\\ \forall v_i \text{ occurring in } S\end{array}$$

$$\frac{\langle S, x : \neg\Box\neg C \mid U\rangle}{\langle S, x : \neg\Box\neg C, y < x, y : C, y : \Box\neg C, S^M_{x\to y} \mid U\rangle \ \langle S, x : \neg\Box\neg C, v_1 < x, v_1 : C, v_1 : \Box\neg C, S^M_{x\to v_1} \mid U\rangle \cdots \langle S, x : \neg\Box\neg C, v_n < x, v_n : C, v_n : \Box\neg C, S^M_{x\to v_n} \mid U\rangle}\ (\Box^-)$$
$$\begin{array}{r}y \text{ new}\\ \text{if } \nexists z \prec x \text{ s.t. } z \equiv_{S,x:\neg\Box\neg C} x \text{ and } \nexists u \text{ s.t. } \{u < x, u : C, u : \Box\neg C, S^M_{x\to u}\} \subseteq S\\ \forall v_i \text{ occurring in } S, x \neq v_i\end{array}$$

Fig. 1. The calculus $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH1}$.

$$\langle S, x : C, x : \neg C \mid U \mid K\rangle\ \text{(Clash)} \quad \langle S, x : \bot \mid U \mid K\rangle\ \text{(Clash)}_{\bot} \quad \langle S, x : \neg\top \mid U \mid K\rangle\ \text{(Clash)}_{\top} \quad \langle S \mid U \mid \emptyset\rangle\ \text{(Clash)}_{\emptyset}$$

$$\langle S, x : \neg\Box\neg C \mid U \mid K\rangle\ \text{(Clash)}_{\Box^-} \quad \text{if } x : \neg\Box\neg C \notin \mathbf{B}^{\Box^-}$$

$$\frac{\langle S, x : \mathbf{T}(C) \mid U \mid K\rangle}{\langle S, x : C, x : \Box\neg C \mid U \mid K\rangle}\ (\mathbf{T}^+)$$

$$\frac{\langle S, x : \neg\mathbf{T}(C) \mid U \mid K\rangle}{\langle S, x : \neg C \mid U \mid K\rangle \qquad \langle S, x : \neg\Box\neg C \mid U \mid K\rangle}\ (\mathbf{T}^-)$$

$$\frac{\langle S \mid U, C \sqsubseteq D^L \mid K\rangle}{\langle S, x : \neg C \sqcup D \mid U, C \sqsubseteq D^{L,x} \mid K\rangle}\ (\sqsubseteq) \quad x \in \mathcal{D}(\mathbf{B}) \text{ and } x \notin L$$

$$\frac{\langle S, x : \forall R.C, x \xrightarrow{R} y \mid U \mid K\rangle}{\langle S, x : \forall R.C, x \xrightarrow{R} y, y : C \mid U \mid K\rangle}\ (\forall^+) \quad \text{if } y : C \notin S$$

$$\frac{\langle S, x : \exists R.C \mid U \mid K\rangle}{\langle S, x \xrightarrow{R} v_1, v_1 : C \mid U \mid K\rangle \qquad \langle S, x \xrightarrow{R} v_2, v_2 : C \mid U \mid K\rangle \cdots \langle S, x \xrightarrow{R} v_n, v_n : C \mid U \mid K\rangle}\ (\exists^+) \quad \forall v_i \in \mathcal{D}(\mathbf{B})$$

$$\frac{\langle S \mid U \mid K\rangle}{\langle S, x : \Box\neg C \mid U \mid K\rangle \qquad \langle S, x : \neg\Box\neg C \mid U \mid K\rangle}\ (cut) \quad \begin{array}{r}\text{if } x : \neg\Box\neg C \notin S \text{ and } x : \Box\neg C \notin S\\ x \in \mathcal{D}(\mathbf{B}) \quad C \in \mathcal{L}_{\mathbf{T}}\end{array}$$

$$\frac{\langle S, x : \neg\Box\neg C \mid U \mid K, x : \neg\Box\neg C\rangle}{\langle S, v_1 : C, v_1 : \Box\neg C, S^M_{x\to v_1}, x : \neg\Box\neg C \mid U \mid K\rangle \ \langle S, v_2 : C, v_2 : \Box\neg C, S^M_{x\to v_2}, x : \neg\Box\neg C \mid U \mid K\rangle \cdots \langle S, v_n : C, v_n : \Box\neg C, S^M_{x\to v_n}, x : \neg\Box\neg C \mid U \mid K\rangle}\ (\Box^-)$$
$$\begin{array}{r}\text{if } \nexists u \text{ s.t. } \{u : C, u : \Box\neg C, S^M_{x\to u}\} \subseteq S\\ \forall v_i \in \mathcal{D}(\mathbf{B}), x \neq v_i\end{array}$$

Fig. 2. The calculus $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH2}$. To save space, we only include the most relevant rules.

A tableau of $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH2}$ is a tree whose nodes are tuples of the form $\langle S \mid U \mid K\rangle$, where S and U are defined as in $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH1}$, whereas K contains formulas of the form $x : \neg\Box\neg C$, with $C \in \mathcal{L}_{\mathbf{T}}$. The basic idea of $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH2}$ is as follows. Given an open branch $\mathbf{B}$ built by $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH1}$ and corresponding to a model $\mathcal{M}^{\mathbf{B}}$ of KB $\cup \{\neg F\}$, $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH2}$ checks whether $\mathcal{M}^{\mathbf{B}}$ is a minimal model of KB by trying to build a model of KB which is preferred to $\mathcal{M}^{\mathbf{B}}$. To this purpose, it keeps track (in K) of the negated box formulas used in $\mathbf{B}$ ($\mathbf{B}^{\Box^-}$) in order to check whether it is possible to build a model of KB containing less negated box formulas.

The rules of $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH2}$ are shown in Figure 2. The tableau built by $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH2}$ closes if it is not possible to build a model smaller than $\mathcal{M}^{\mathbf{B}}$, it remains open otherwise. Since by Definition 3 two models can be compared only if they have the same domain, $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH2}$ tries to build an open branch containing all the labels appearing in $\mathbf{B}$, i.e. those in $\mathcal{D}(\mathbf{B})$. To this aim, the dynamic rules use labels in $\mathcal{D}(\mathbf{B})$ instead of introducing new ones in their conclusions. The rule $(\sqsubseteq)$ is applied to *all the labels of* $\mathcal{D}(\mathbf{B})$ (and not only to those appearing in the branch). The rule $(\Box^-)$ is applied to a node $\langle S, x : \neg\Box\neg C \mid U \mid K, x : \neg\Box\neg C\rangle$, that is to say when the negated box formula $x : \neg\Box\neg C$ also belongs to the open branch **B**. Also in this case, the rule introduces a branch on the choice of the individual $v_i \in \mathcal{D}(\mathbf{B})$ to be used in the conclusion. In case a tableau node has the form $\langle S, x : \neg\Box\neg C \mid U \mid K\rangle$, and $x : \neg\Box\neg C \notin \mathbf{B}^{\Box^-}$, then $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ detects a clash, called $(\text{Clash})_{\Box^-}$: this corresponds to the situation where $x : \neg\Box\neg C$ does not belong to **B**, while the model corresponding to the branch being built contains $x : \neg\Box\neg C$, and hence is *not* preferred to the model represented by **B**. The calculus $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ also contains the clash condition $(\text{Clash})_{\emptyset}$. Since each application of $(\Box^-)$ removes the negated box formulas $x : \neg\Box\neg C$ from the set K, when K is empty all the negated boxed formulas occurring in **B** also belong to the current branch. In this case, the model built by $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ satisfies the same set of $x : \neg\Box\neg C$ (for all individuals) as **B** and, thus, it is not preferred to the one represented by **B**.

Let KB be a knowledge base whose corresponding constraint system is $\langle S \mid U\rangle$. Let F be a query and let S' be the set of constraints obtained by adding to S the constraint corresponding to $\neg F$. $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ is *sound and complete* in the following sense: an open branch **B** built by $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ for $\langle S' \mid U\rangle$ is satisfiable in a minimal model of KB iff the tableau in $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ for $\langle S \mid U \mid \mathbf{B}^{\Box^-}\rangle$ is closed.

The termination of $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ is ensured by the fact that dynamic rules make use of labels belonging to $\mathcal{D}(\mathbf{B})$, which is finite, rather than introducing "new" labels in the tableau. Also, it is possible to show that the problem of verifying that a branch **B** represents a minimal model for KB in $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ is in NP in the size of **B**.

The overall procedure $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$ is defined as follows:

Definition 5. *Let KB be a knowledge base whose corresponding constraint system is $\langle S \mid U\rangle$. Let F be a query and let S' be the set of constraints obtained by adding to S the constraint corresponding to $\neg F$. The calculus $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$ checks whether a query F can be minimally entailed from a* KB *by means of the following procedure:*

- *the calculus $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ is applied to $\langle S' \mid U\rangle$;*
- *if, for each branch* **B** *built by $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$, either: (i)* **B** *is closed or (ii) the tableau built by the calculus $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ for $\langle S \mid U \mid \mathbf{B}^{\Box^-}\rangle$ is open, then the procedure says* YES *else the procedure says* NO

In [13] we have shown that $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$ is a sound and complete decision procedure for verifying if KB $\models^{\mathcal{L}_\mathbf{T}}_{\mathcal{ALC}+\mathbf{T}_{min}} F$, and that the problem is in CO-NEXP$^{\text{NP}}$.

4 Design of `DysToPic`

In this section we present `DysToPic`, a multi-engine theorem prover for reasoning in $\mathcal{ALC} + \mathbf{T}_{min}$. `DysToPic` is a SICStus Prolog implementation of the tableaux calculi $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$ introduced in the previous section, wrapped by a Java interface which relies on the Java RMI APIs for the distribution of the computation. The system is designed for scalability and based on a "worker/employer" paradigm: the computational burden for the "employer" can be spread among an arbitrarily high number of "workers" which operate in complete autonomy, so that they can be either deployed on a single machine or on a computer grid.

The basic idea underlying DysToPic is as follows: there is no need for the first phase of the calculus to wait for the result of one elaboration of the second phase on an open branch, before generating another candidate branch. Indeed, in order to prove whether a query F entails from a KB, the first phase can be executed on a machine; every time that a branch remains open after the first phase, the execution of the second phase for this branch can be performed in parallel, on a different machine. Meanwhile, the main machine (worker), instead of waiting for the termination of the second phase on that branch, can carry on with the computation of the first phase (potentially generating other branches). If a branch remains open in the second phase, then F is not minimally entailed from KB (we have found a counterexample), so the computation process can be interrupted early.

4.1 The Whole Architecture

In order to describe the architecture of DysToPic we refer to the *worker-employer* metaphor. The system is characterized by: (i) a single *employer*, which is in charge of verifying the query and yielding the final result. It also implements the first phase of the calculus and uses $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ to generate branches: the ones that it cannot close (representing candidate models of KB $\cup\{\neg F\}$), it passes to a *worker*; (ii) an unlimited number of *workers*, which use $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ to evaluate the models generated by the *employer*; (iii) a *repository*, which stores all the answers coming from the *workers*.
First, each worker registers to the employer. When checking whether KB $\models_{\mathcal{ALC}+\mathbf{T}_{min}}$ F, the employer executes $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$. If the employer needs to check whether an open branch generated by the first phase represents a minimal model of the KB, then it delegates the execution of the second phase to one of the registered workers, and consequently proceeds with its computation on other branches generated in the first phase. When a worker terminates its execution, it reports its result to the repository.

If every branch has been processed and each worker has answered affirmatively, i.e. each tableaux built in the second phase by $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ is open, the employer can conclude that KB $\models_{\mathcal{ALC}+\mathbf{T}_{min}}$ F. Otherwise, the employer can conclude the proof as soon as the first negative answer comes into the repository, since (at least) a worker found a closed tableaux in $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$ for an open branch (candidate model) generated by the employer, in this case we have that KB $\not\models_{\mathcal{ALC}+\mathbf{T}_{min}}$ F. It is worth noticing that the employer has to keep a continuous dialogue with the repository.

The library `se.sics.jasper` is used in order to combine Java and SICStus Prolog to decouple the two phases of the calculus. In detail, the employer handles the query in `Employer.java`, a piece of Java code which presents it to `alct1.pl`, the Prolog core implementing $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$. Every time that an open branch is generated, `alct1.pl` invokes `Phase1RMIStub.java`, another piece of Java code which will send it to the correct worker. Workers will then have to process the open branches with $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$, which is implemented in `alct2.pl`.

Concurrency is the main goal of our implementation, since we want the execution of the first phase of the calculus to be independent from the second one. Java natively supports concurrency via multithreading. The employer uses a separate thread (implemented in `Phase1Thread.java`) to perform the current invocation of $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$ on a query, while its main thread polls the repository waiting for termination (the procedure can be stopped when the first counterexample is found, even if not all of the branches have been explored). During the execution of $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$, every time that the employer wants to ask a worker to verify a branch, a new thread is spawned. The worker itself makes use of threads: its main thread simply enqueues each request coming from the employer and spawns a new thread which performs $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$.

4.2 The Implementation of the Tableaux Calculi

Concerning the implementation of the tableaux calculi $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$, each machine of the system runs a SICStus Prolog implementation which is strongly related to the implementation of the calculi given by `PreDeLo 1.0`, introduced in [11]. The implementation is inspired by the "lean" methodology of leanT^AP, even if it does not follow its style in a rigorous manner. The program comprises a set of clauses, each one implementing a rule or axiom of the tableau calculi. The proof search is provided for free by the mere depth-first search mechanism of Prolog, without any additional mechanism.

`DysToPic` comprises two main predicates, called `prove` and `prove_phase2`, implementing, respectively, the first and the second phase of the tableau calculi.

Phase 1: the `prove` predicate. Concerning the first phase of the calculi, executed by the employer, `DysToPic` represents a tableaux node $\langle S \mid U \rangle$ with two Prolog lists: `S` and `U`. Elements of `S` are either pairs `[X, F]`, representing formulas of the form $x : F$, or triples of the form either `[X,R,Y]` or `[X,<,Y]`, representing either roles $x \xrightarrow{R} y$ or the preference relation $x < y$, respectively. Elements of `U` are pairs of the form `[[C inc D],L]`, representing $C \sqsubseteq D^L \in U$ described in Section 3.1.

The calculi $\mathcal{TAB}_{min}^{\mathcal{ALC}+\mathbf{T}}$ are implemented by a top-level predicate

```
prove(+ABox,+TBox,[+X,+F],-Tree).
```

This predicate succeeds if and only if the query $x : F$ is minimally entailed from the KB represented by `TBox` and `ABox`. When the predicate succeeds, then the output term `Tree` matches a Prolog term representing the closed tableaux found by the prover. The top-level predicate `prove/4` invokes a second-level one:

```
prove(+S,+U,+Lt,+Labels,+ABOX,-Tree)
```

having 6 arguments. In detail, `S` corresponds to `ABox` enriched by the negation of the query $x : F$, whereas `Lt` is a list corresponding to the set of concepts $\mathcal{L}_{\mathbf{T}}$. `Labels` is the set of labels belonging to the current branch, whereas `ABOX` is used to store the initial ABox (i.e. without the negation of the query) in order to eventually invoke the second phase on it, in order to look for minimal models of the initial KB.

Each clause of the `prove/6` predicate implements an axiom or rule of the calculus $\mathcal{TAB}_{PH1}^{\mathcal{ALC}+\mathbf{T}}$. To search a closed tableaux for $\langle S \mid U \rangle$, `DysToPic` proceeds as follows. First of all, if $\langle S \mid U \rangle$ is a clash, the goal will succeed immediately by using one of the clauses implementing axioms. As an example, the following clause implements (Clash):

```
prove(S,U,_,_,_,tree(clash)):-
      member([X,C],S),member([X, neg C],S),!.
```

If $\langle S \mid U \rangle$ is not an instance of the axioms, then the first applicable rule will be chosen, e.g. if `S` contains an intersection `[X,C and D]`, then the clause implementing the $(\sqcap^+)$ rule will be chosen, and `DysToPic` will be recursively invoked on its unique conclusion. `DysToPic` proceeds in a similar way for the other rules. The ordering of the clauses is such that the application of the dynamic rules is postponed as much as possible: this implements the strategy ensuring the termination of the calculi described in the previous section. As an example, the clause implementing $(\mathbf{T}^+)$ is as follows:

```
1. prove(S,U,Lt,Labels,ABOX,tree(...,Tree)):-member([X,ti C],S),
2.    (\+(member([X,C],S)); \+(member([X, box neg C],S))),!,
3.    prove([[X,C]|[[X, box neg C]|S]],U,Lt,Labels,ABOX,Tree),!.
```

In line 1, the standard Prolog predicate `member` is used in order to find a formula of the form $x : \mathbf{T}(C)$ in the list `S`. In line 2, the side conditions on the applicability of such a rule are checked: the rule can be applied if either $x : C$ or $x : \Box\neg C$ do not belong to `S`. In line 3 `DysToPic` is recursively invoked on the unique conclusion of the rule, in which $x : C$ and $x : \Box\neg C$ are added to the list `S`. The last clause of `prove` is:

```
prove(...) :- ... , jasper_call(JVM,
    method('employer/Phase1RMIStub','solveViaRMI',[static]),...,
    solve_via_rmi(NextWorkerName, 'toplevelphase2(...)' ) ),!.
```

invoked when no other clauses are applicable. In this case, the branch built by the employer represents a model for the initial set of formulas, then the `toplevelphase2` predicate is invoked on a worker in order to check whether such a model is minimal for KB.

Phase 2: the `prove_phase2` predicate. Given an open branch built by the first phase, the predicate `toplevelphase2` is invoked on a worker. It first applies an optimization preventing useless applications of $(\sqsubseteq)$, then it invokes the predicate

`prove_phase2(+S,+U,+Lt,+K,+Bb,+Db).`

`S` and `U` contain the initial KB (without the query), whereas `K`, `Bb` and `Db` are Prolog lists representing K, $\mathbf{B}^{\Box^-}$ and $\mathcal{D}(\mathbf{B})$ as described in Section 3.2. `Lt` is as for `prove/6`.

Also in this case, each clause of `prove_phase2` implements an axiom or rule of the calculi $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$. To search for a closed tableaux, `DysToPic` first checks whether the current node $\langle S \mid U \mid K \rangle$ is a clash. otherwise the first applicable rule will be chosen, and `DysToPic` will be recursively invoked on its conclusions. As an example, the clause implementing $(\mathbf{T}^+)$ is as follows:

```
prove_phase2(S,U,Lt,K,Bb,Db) :- select([X,ti C],S,S1),
       prove_phase2([[X,C]|[[X,box neg C]|S1]],U,Lt,K,Bb,Db),!.
```

Notice that, according to the calculus $\mathcal{TAB}_{PH2}^{\mathcal{ALC}+\mathbf{T}}$, the principal formula to which the rule is applied is removed from the current node: to this aim, the SICStus Prolog predicate `select` is used rather than `member`.

4.3 Preliminary Performance Testing of `DysToPic`

We have made a preliminary attempt to show how `DysToPic` performs, especially in comparison with its predecessor `PreDeLo 1.0`. The performances of `DysToPic` are promising. We have tested both the provers by running SICStus Prolog 4.1.1 on Ubuntu

14.04.1 64 bit machines. Concerning `DysToPic`, we have tested it on 4 machines, namely: 1. a desktop PC with an Intel Core i5-3570K CPU (3.4-3.8GHz, 4 cores, 4 threads, 8GB RAM); 2. a desktop PC with an Intel Pentium G2030 CPU (3.0GHz, 2 cores, 2 threads, 4GB RAM); 3. a Lenovo X220 laptop with an Intel Core i7-2640M CPU (2.8-3.5GHz, 2 cores, 4 threads, 8GB RAM); 4. a Lenovo X230 laptop with an Intel Core i7-3520M CPU (2.9-3.6GHz, 2 cores, 4 threads, 8GB RAM).

We have performed two kinds of tests. On the one hand, we have randomly generated KBs with different sizes (from 10 to 100 ABox formulas and TBox inclusions) as well as different numbers of named individuals: in less than 10 seconds, both `DysToPic` and `PreDeLo 1.0` are able to answer in more than the 75% of tests. Notice that, as far as we know, it does not exist a set of acknowledged benchmarks for defeasible DLs. On the other hand, we have tested the two theorem provers on specific examples. As expected, `DysToPic` is better in than the competitor in answering that a query F is not minimally entailed from a given KB. Surprisingly enough, its performances are better than the ones of `PreDeLo 1.0` also in case the provers conclude that F follows from KB, as in the following example:

Example 1. Given TBox=$\{\mathbf{T}(\mathit{Student}) \sqsubseteq \neg \mathit{IncomeTaxPayer}, \mathit{WorkingStudent} \sqsubseteq \mathit{Student}, \mathbf{T}(\mathit{WorkingStudent}) \sqsubseteq \mathit{IncomeTaxPayer}\}$ and ABox=$\{\mathit{Student}(\mathit{mario})$, $\mathit{WorkingStudent}(\mathit{mario})$, $\mathit{Tall}(\mathit{mario})$, $\mathit{Student}(\mathit{carlo})$, $\mathit{WorkingStudent}(\mathit{carlo})$, $\mathit{Tall}(\mathit{carlo})$, $\mathit{Student}(\mathit{giuseppe})$, $\mathit{WorkingStudent}(\mathit{giuseppe})$, $\mathit{Tall}(\mathit{giuseppe})\}$, we have tested both the provers in order to check whether $\mathit{IncomeTaxPayer}(\mathit{mario})$ is minimally entailed from KB=(TBox,ABox). This query generates 1090 open branches in $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH1}$, each one requiring the execution of $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{PH2}$. `PreDeLo 1.0` answers in 370 seconds, whereas `DysToPic` answers in 210 seconds if only two machines are involved (employer + one worker). If 4 workers are involved, `DysToPic` only needs 112 seconds to conclude its computation.

Example 1 witnesses that the advantages obtained by distributing the computation justify the overhead introduced by the machinery needed for such distribution.

5 Conclusions

We have introduced `DysToPic`, a multi-engine theorem prover implementing tableaux calculi for reasoning in preferential DL $\mathcal{ALC} + \mathbf{T}_{min}$. `DysToPic` implements a distributed version of the tableaux calculus $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{min}$ introduced in [13], exploiting the fact that the two phases characterizing such a calculus can be computed in parallel.

We aim at extending `DysToPic` to the *lightweight* DLs of the DL-Lite and $\mathcal{EL}$ family. Despite their relatively low expressivity, they are relevant for several applications, in particular in the bio-medical domain. Extensions of $\mathcal{EL}^{\perp}$ and of DL-Lite$_{core}$ with the typicality operator $\mathbf{T}$ have been proposed in [10], where it has also been shown that minimal entailment is in Π^p_2 (for $\mathcal{EL}^{\perp}$, if restricted to a specific fragment). Tableaux calculi performing a two phases computation, similar to $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{min}$, have been proposed in [9, 8].

This research is partially supported by INDAM- GNCS Project 2015 "Logiche descrittive e ragionamento non monotono".

References

1. Baader, F., Hollunder, B.: Priorities on defaults with prerequisites, and their application in treating specificity in terminological default logic. Journal of Automated Reasoning (JAR) 15(1), 41–68 (1995)
2. Beckert, B., Posegga, J.: leantap: Lean tableau-based deduction. Journal of Automated Reasoning (JAR) 15(3), 339–358 (1995)
3. Bonatti, P.A., Lutz, C., Wolter, F.: DLs with Circumscription. In: KR. pp. 400–410 (2006)
4. Bonatti, P.A., Faella, M., Sauro, L.: Defeasible inclusions in low-complexity dls: Preliminary notes. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009). pp. 696–701 (2009)
5. Casini, G., Straccia, U.: Rational closure for defeasible description logics. In: Janhunen, T., Niemelä, I. (eds.) Logics in Artificial Intelligence - Proceedings of the12th European Conference (JELIA 2010). Lecture Notes in Computer Science (LNCS), vol. 6341, pp. 77–90. Springer (2010)
6. Donini, F.M., Nardi, D., Rosati, R.: Description logics of minimal knowledge and negation as failure. ACM Transactions on Computational Logics (ToCL) 3(2), 177–225 (2002)
7. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: ALC+T: a preferential extension of description logics. Fundamenta Informaticae 96, 341–372 (2009)
8. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: A tableau calculus for a nonmonotonic extension of $\mathcal{EL}^{\perp}$. In: Brünnler, K., Metcalfe, G. (eds.) Proceedings of the 20th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2011). Lecture Notes in Artificial Intelligence (LNAI), vol. 6793, pp. 180–195. Springer-Verlag, Bern, Switzerland (July 2011)
9. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: A tableau calculus for a nonmonotonic extension of the Description Logic $DL-Lite_{core}$. In: Pirrone, R., Sorbello, F. (eds.) AI*IA 2011: Artificial Intelligence Around Man and Beyond - XIIth International Conference of the Italian Association for Artificial Intelligence, Palermo, Italy, September 15-17, 2011. Proceedings. Lecture Notes in Artificial Intelligence (LNAI), vol. 6934, pp. 164–176. Springer-Verlag, Palermo, Italy (Sptember 2011)
10. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Reasoning about typicality in low complexity DLs: the logics $\mathcal{EL}^{\perp}\mathbf{T}_{min}$ and *DL-Lite*$_c\mathbf{T}_{min}$. In: Walsh, T. (ed.) Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011). pp. 894–899. IOS Press, Barcelona, Spain (2011)
11. Giordano, L., Gliozzi, V., Jalal, A., Olivetti, N., Pozzato, G.L.: Predelo 1.0: a theorem prover for preferential description logics. In: Baldoni, M., Baroglio, C., Boella, G., Micalizio, R. (eds.) Proceedings of AI*IA 2013. Lecture Notes in Artificial Intelligence LNAI, vol. 8249, pp. 60 – 72. Springer, Torino (December 2013)
12. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Preferential vs Rational Description Logics: which one for Reasoning About Typicality? . In: Coelho, H., Studer, R., Wooldridge, M. (eds.) Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010). FAIA (Frontiers in Artificial Intelligence and Applications), vol. 215, pp. 1069 – 1070. IOS Press, Lisbon, Portugal (August 2010)
13. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: A NonMonotonic Description Logic for Reasoning About Typicality. Artificial Intelligence 195, 165 – 202 (2013)
14. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence 44(1-2), 167–207 (1990)
15. Lehmann, D., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence 55(1), 1–60 (1992)