An Algorithm for Adapting Cases Represented in an Expressive Description Logic

Julien Cojan and Jean Lieber

UHP-Nancy 1 – LORIA (UMR 7503 CNRS-INPL-INRIA-Nancy 2-UHP)
BP 239, 54506 Vandœuvre-lès-Nancy, France
{Julien.Cojan, Jean.Lieber}@loria.fr

Abstract. This paper presents an algorithm of adaptation for a case-based reasoning system with cases and domain knowledge represented in the expressive description logic $\mathcal{ALC}$. The principle is to first pretend that the source case to be adapted solves the current target case. This may raise some contradictions with the specification of the target case and with the domain knowledge. The adaptation consists then in repairing these contradictions. This adaptation algorithm is based on an extension of the classical tableau method used for deductive inference in $\mathcal{ALC}$.

Keywords: adaptation, description logic, $\mathcal{ALC}$, tableau algorithm

1 Introduction

Adaptation is a step of some case-based reasoning (CBR) systems that consists in modifying a source case in order to suit a new situation, the target case. An approach to adaptation consists in using a belief revision operator, i.e., an operator that modifies minimally a set of beliefs in order to be consistent with some actual knowledge [1]. The idea is to consider the belief "The source case solves the target case" and then to revise it with the constraints given by the target case and the domain knowledge. This has been studied for cases represented in propositional logic in [9]. Then, it has been studied in a more expressive formalism, including numerical constraints and after that extended to the combination of cases in this formalism [3].

In this paper, this approach to adaptation is studied for cases represented in an expressive description logic (DL), namely $\mathcal{ALC}$. The choice of DLs as formalisms for CBR can be motivated in several ways. First, they extend the classical attribute-value formalisms, often used in CBR (see, e.g., [8]) and they are similar to the formalism of memory organisation packets (MOPs) used in early CBR applications [10]. More generally, they are designed as trade-offs between expressibility and practical tractability. Second, they have a well-defined semantics and have been systematically investigated for several decades, now. Third, many efficient implementations are freely available, offering services that can be used for CBR systems, in particular for case retrieval and case base organisation.

The rest of the paper is organised as follows. Section 2 presents the DL $\mathcal{ALC}$, together with the tableau algorithm, at the basis of its deductive inferences for most current implementation. An example is presented in this section, for illustrating notions

that are rather complex for a reader not familiar with DLs. This tableau algorithm is extended for performing an adaptation process, as shown in section 3. Section 4 discusses our contribution and relates it to other research on the use of DLs for CBR. Section 5 concludes the paper and presents some future work.

2 The Description Logic $\mathcal{ALC}$

Description logics [2] form a family of classical logics that are equivalent to decidable fragments of first-order logic (FOL). They have a growing importance in the field of knowledge representation. $\mathcal{ALC}$ is the simplest of *expressive DLs*, i.e., DLs extending propositional logic. The example presented in this section is about cooking, in the spirit of the computer cooking contest, but sticks to the adaptation of the ingredient list.

2.1 Syntax

Representation entities of $\mathcal{ALC}$ are concepts, roles, instances, and formulas.

A *concept*, intuitively, represents a subset of the interpretation domain. A concept is either an *atomic concept* (i.e., a concept name), or a conceptual expression of one of the forms $\top$, $\bot$, $\neg\mathtt{C}$, $\mathtt{C} \sqcap \mathtt{D}$, $\mathtt{C} \sqcup \mathtt{D}$, $\exists\mathtt{r}.\mathtt{C}$, and $\forall\mathtt{r}.\mathtt{C}$, where C and D are concepts (either atomic or not) and r is a role. A concept can be mapped into a FOL formula with one free variable x. For example, the concept

$$\mathtt{Tart} \sqcap \exists\mathtt{ing}.\mathtt{Apple} \sqcap \exists\mathtt{ing}.\mathtt{Pastry} \sqcap \forall\mathtt{ing}.\neg\mathtt{Cinnamon} \tag{1}$$

can be mapped to the first-order logic formula

$$\begin{aligned}\mathtt{Tart}(x) \wedge (\exists y, \mathtt{ing}(x,y) \wedge \mathtt{Apple}(y)) \wedge (\exists y, \mathtt{ing}(x,y) \wedge \mathtt{Pastry}(y))\\ \wedge (\forall y, \mathtt{ing}(x,y) \Rightarrow \neg\mathtt{Cinnamon}(y))\end{aligned}$$

A *role*, intuitively, represents a binary relation on the interpretation domain. Roles in $\mathcal{ALC}$ are atomic: i.e., role names. Their counterpart in FOL are binary predicates. The role appearing in (1) is ing.

An *instance*, intuitively, represents an element of the interpretation domain. Instances in $\mathcal{ALC}$ are atomic: i.e., instance names. Their counterpart in FOL are constants.

There are four types of formulas in $\mathcal{ALC}$ (followed by their meaning): (1) $\mathtt{C} \sqsubseteq \mathtt{D}$ (C is more specific than D), (2) $\mathtt{C} \equiv \mathtt{D}$ (C and D are equivalent concepts), (3) $\mathtt{C}(\mathtt{a})$ (a is an instance of C), and (4) $\mathtt{r}(\mathtt{a},\mathtt{b})$ (r relates a to b), where C and D are concepts, a and b are instances, and r is a role. Formulas of types (1) and (2) are called *terminological formulas*. Formulas of types (3) and (4) are called *assertional formulas*, or *assertions*.

An $\mathcal{ALC}$ knowledge base KB is a set of $\mathcal{ALC}$ formulas. The *terminological box* (or *TBox*) of KB is the set of its terminological formulas. The *assertional box* (or *ABox*) of KB is the set of its assertions.

For example, the following TBox represents the domain knowledge (DK) of our example (with the comments giving the meaning):

$$\begin{aligned}\mathtt{DK} = \{&\mathtt{Apple} \sqsubseteq \mathtt{PomeFruit}, && \textit{An apple is a pome fruit.}\\ &\mathtt{Pear} \sqsubseteq \mathtt{PomeFruit}, && \textit{A pear is a pome fruit.}\\ &\mathtt{PomeFruit} \sqsubseteq \mathtt{Apple} \sqcup \mathtt{Pear}\} && \textit{A pome fruit is either an apple or a pear.}\end{aligned} \tag{2}$$

Note that the last formula is a simplification: actually, there are other pome fruits than apples and pears.

In our running example, the only cases considered are the source and target cases. They are represented in the ABox:

$$\{\texttt{Source}(\sigma), \texttt{Target}(\theta)\} \tag{3}$$

$$\text{with } \begin{cases} \texttt{Source} = \texttt{Tart} \sqcap \exists \texttt{ing}.\texttt{Pastry} \sqcap \exists \texttt{ing}.\texttt{Apple} \\ \texttt{Target} = \texttt{Tart} \sqcap \forall \texttt{ing}.\neg \texttt{Apple} \end{cases} \tag{4}$$

Thus, the source case is represented by the instance σ, which is a tart with the types of ingredients pastry and apple. The target case is represented by the instance θ specifying that a tart without apple is requested.

Reusing the source case without adaptation for the target case amounts to add the assertion $\texttt{Source}(\theta)$. However this may lead to contradictions like here between $\exists \texttt{ing}.\texttt{Apple}(\theta)$ and $\forall \texttt{ing}.\neg \texttt{Apple}(\theta)$: the source case needs to be adapted before being applied to the target case.

2.2 Semantics

An interpretation is a pair $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$ where $\Delta_\mathcal{I}$ is a non empty set (the *interpretation domain*) and where $\cdot^\mathcal{I}$ maps a concept $\texttt{C}$ into a subset $\texttt{C}^\mathcal{I}$ of $\Delta_\mathcal{I}$, a role $\texttt{r}$ into a binary relation $\texttt{r}^\mathcal{I}$ over $\Delta_\mathcal{I}$ (for $x, y \in \Delta_\mathcal{I}$, x is related to y by $\texttt{r}^\mathcal{I}$ is denoted by $(x, y) \in \texttt{r}^\mathcal{I}$), and an instance $\texttt{a}$ into an element $\texttt{a}^\mathcal{I}$ of $\Delta_\mathcal{I}$.

Given an interpretation $\mathcal{I}$, the different types of conceptual expressions are interpreted as follows:

$$\begin{aligned} \top^\mathcal{I} &= \Delta_\mathcal{I} & \bot^\mathcal{I} &= \emptyset \\ (\neg \texttt{C})^\mathcal{I} &= \Delta_\mathcal{I} \setminus \texttt{C}^\mathcal{I} & & \\ (\texttt{C} \sqcap \texttt{D})^\mathcal{I} &= \texttt{C}^\mathcal{I} \cap \texttt{D}^\mathcal{I} & (\texttt{C} \sqcup \texttt{D})^\mathcal{I} &= \texttt{C}^\mathcal{I} \cup \texttt{D}^\mathcal{I} \\ (\exists \texttt{r}.\texttt{C})^\mathcal{I} &= \{x \in \Delta_\mathcal{I} \mid \exists y, (x, y) \in \texttt{r}^\mathcal{I} \text{ and } y \in \texttt{C}^\mathcal{I}\} & & \\ (\forall \texttt{r}.\texttt{C})^\mathcal{I} &= \{x \in \Delta_\mathcal{I} \mid \forall y, \text{ if } (x, y) \in \texttt{r}^\mathcal{I} \text{ then } y \in \texttt{C}^\mathcal{I}\} & & \end{aligned}$$

For example, if $\texttt{Tart}^\mathcal{I}$, $\texttt{Apple}^\mathcal{I}$, $\texttt{Pastry}^\mathcal{I}$, and $\texttt{Cinnamon}^\mathcal{I}$ denote the sets of tarts, apples, pastries, and cinnamon, and if $\texttt{ing}^\mathcal{I}$ denotes the relation "has the ingredient", then the concept of equation (1) denotes the set of the tarts with apples and pastries, but without cinnamon.

Given a formula f and an interpretation $\mathcal{I}$, "$\mathcal{I}$ satisfies f" is denoted by $\mathcal{I} \models f$. A model of f is an interpretation $\mathcal{I}$ satisfying f. The semantics of the four types of formulas is as follows:

$$\begin{aligned} \mathcal{I} &\models \texttt{C} \sqsubseteq \texttt{D} & &\text{if } \texttt{C}^\mathcal{I} \subseteq \texttt{D}^\mathcal{I} \\ \mathcal{I} &\models \texttt{C} \equiv \texttt{D} & &\text{if } \texttt{C}^\mathcal{I} = \texttt{D}^\mathcal{I} \\ \mathcal{I} &\models \texttt{C}(\texttt{a}) & &\text{if } \texttt{a}^\mathcal{I} \in \texttt{C}^\mathcal{I} \\ \mathcal{I} &\models \texttt{r}(\texttt{a}, \texttt{b}) & &\text{if } (\texttt{a}^\mathcal{I}, \texttt{b}^\mathcal{I}) \in \texttt{r}^\mathcal{I} \end{aligned}$$

Given a knowledge base KB and an interpretation $\mathcal{I}$, $\mathcal{I}$ satisfies KB –denoted by $\mathcal{I} \models$ KB– if $\mathcal{I} \models f$ for each $f \in$ KB. A *model* of KB is an interpretation satisfying KB. A knowledge base KB entails a formula f –denoted by KB $\models f$– if every model of KB is a model of f. A tautology is a formula f satisfied by any interpretation. "f is a tautology" is denoted by $\models f$. Two knowledge bases are said to be equivalent if every model of one of them is a model of the other one and vice-versa.

2.3 Inferences

Let KB be a knowledge base. Some classical inferences on $\mathcal{ALC}$ consist in checking if KB $\models f$, for some formula f. For instance, checking if KB $\models$ C $\sqsubseteq$ D is called the *subsumption test*: it tests whether, according to the knowledge base, the concept C is more specific than the concept D, and thus is useful for organising concepts in hierarchies (e.g., index hierarchies of CBR systems).

The *concept classification* consists, given a concept C, in finding the atomic concepts A appearing in KB such that KB $\models$ C $\sqsubseteq$ A (the subsumers of C) and the atomic concepts B appearing in KB such that KB $\models$ B $\sqsubseteq$ C (the subsumees of C). The *instance classification* consists, given an instance a, in finding the atomic concepts A appearing in KB such that KB $\models$ A(a). These two inferences can be used for case retrieval in a CBR system.

The *ABox satisfiability* consists in checking, given an ABox, whether there exists a model of this ABox, given a knowledge base KB. Some other important inferences can be reduced to it, for instance:

$$\text{KB} \models \text{C} \sqsubseteq \text{D} \quad \text{iff} \quad \{(\text{C} \sqcap \neg \text{D})(\text{a})\} \text{ is not satisfiable, given KB}$$

where a is a new instance (not appearing neither in C, nor in KB). ABox satisfiability is also used to detect contradictions, e.g. the one mentioned at the end of section 2.1. It can be computed thanks to the most popular inference mechanism for $\mathcal{ALC}$ presented in next section.

2.4 A classical deduction procedure in $\mathcal{ALC}$: the tableau method

Let KB be a knowledge base, $\mathcal{T}_0$, be the TBox of KB and $\mathcal{A}_0$, be the ABox of KB. The procedure aims at testing whether $\mathcal{A}_0$ is satisfiable or not, given KB.

Preprocessing. The first step of the preprocessing consists in substituting $\mathcal{T}_0$ by an equivalent $\mathcal{T}_0'$ of the form $\{\top \sqsubseteq \text{K}\}$, for some concept K. This can be done by first, substituting each formula C $\equiv$ D by two formulas C $\sqsubseteq$ D and D $\sqsubseteq$ C. The resulting TBox is of the form $\{\text{C}_i \sqsubseteq \text{D}_i\}_{1 \leq i \leq n}$ and it can be shown that it is equivalent to $\{\top \sqsubseteq \text{K}\}$, with

$$\text{K} = (\neg \text{C}_1 \sqcup \text{D}_1) \sqcap \ldots \sqcap (\neg \text{C}_n \sqcup \text{D}_n)$$

The second step of the preprocessing is to put $\mathcal{T}_0$ and $\mathcal{A}_0$ under negative normal form (NNF), i.e., by substituting each concept appearing in them by an equivalent concept such that the negation sign $\neg$ appears only in front of an atomic concept. It is always

possible to do so, by applying, as long as possible, the following equivalences (from left to right):

$$\begin{array}{ccc} \neg\top \equiv \bot & \neg\bot \equiv \top & \neg\neg \mathtt{C} \equiv \mathtt{C} \\ \neg(\mathtt{C} \sqcap \mathtt{D}) \equiv \neg\mathtt{C} \sqcup \neg\mathtt{D} & \neg(\mathtt{C} \sqcup \mathtt{D}) \equiv \neg\mathtt{C} \sqcap \neg\mathtt{D} & \\ \neg\exists \mathtt{r}.\mathtt{C} \equiv \forall \mathtt{r}.\neg\mathtt{C} & \neg\forall \mathtt{r}.\mathtt{C} \equiv \exists \mathtt{r}.\neg\mathtt{C} & \end{array}$$

For example, the concept $\neg(\forall \mathtt{r}.(\neg\mathtt{A} \sqcup \exists \mathtt{s}.\mathtt{B}))$ is equivalent to the following concept under NNF: $\exists \mathtt{r}.(\mathtt{A} \sqcap \forall \mathtt{s}.\neg\mathtt{B})$.

The TBox of DK given in equation (2) is equivalent to $\{\top \sqsubseteq \mathtt{K}\}$ under NNF with

$$\begin{aligned} \mathtt{K} = {} & (\neg\mathtt{Apple} \sqcup \mathtt{PomeFruit}) \sqcap (\neg\mathtt{Pear} \sqcup \mathtt{PomeFruit}) \\ & \sqcap (\neg\mathtt{PomeFruit} \sqcup \mathtt{Apple} \sqcup \mathtt{Pear}) \end{aligned}$$

(technically, to obtain this concept, the equivalence $\mathtt{C} \sqcup \bot \equiv \mathtt{C}$ has also been used).

Main process. Given $\mathcal{T}_0 = \{\top \sqsubseteq \mathtt{K}\}$ a TBox and $\mathcal{A}_0$ an ABox, both under NNF, the tableau method handles sets of ABoxes, starting with the singleton $\mathcal{D}_0 = \{\mathcal{A}_0^{\mathtt{K}}\}$, with

$$\mathcal{A}_0^{\mathtt{K}} = \mathcal{A}_0 \cup \{\mathtt{K}(\mathtt{a}) \mid \mathtt{a} \text{ is an instance appearing in } \mathcal{A}_0\}$$

Such a set of ABoxes $\mathcal{D}$ is to be interpreted as a disjunction: $\mathcal{D}$ is satisfiable iff at least one $\mathcal{A} \in \mathcal{D}$ is satisfiable.

Each further step consists in transforming the current set of ABoxes $\mathcal{D}$ into another one $\mathcal{D}'$, applying some transformation rules on ABoxes: when a rule ϱ, applicable on an ABox $\mathcal{A} \in \mathcal{D}$, is selected by the process, then $\mathcal{D}' = (\mathcal{D} \setminus \{\mathcal{A}\}) \cup \{\mathcal{A}^1, \ldots, \mathcal{A}^p\}$ where the $\mathcal{A}^i$ are obtained by applying ϱ on $\mathcal{A}$ (see further, for the description of the rules).

The process ends when no transformation rule is applicable.

An ABox is *closed* when it contains a *clash*, i.e. an obvious contradiction given by two assertions of the form $\mathtt{A}(\mathtt{a})$ and $(\neg\mathtt{A})(\mathtt{a})$.

Therefore, a closed ABox is unsatisfiable. An *open* ABox is a non-closed ABox.

An ABox is *complete* if no transformation rule can be applied on it.

Let $\mathcal{D}_{\text{end}}$ be the set of ABoxes at the end of the process, i.e. when each $\mathcal{A} \in \mathcal{D}$ is complete. It has been proven (see, e.g., [2]) that, with the transformation rules presented below the process always terminates, and $\mathcal{A}_0$ is satisfiable given $\mathcal{T}_0$ iff $\mathcal{D}_{\text{end}}$ contains at least one open ABox.

The transformation rules. There are four transformations rules for the tableau method applied to $\mathcal{ALC}$: $\longrightarrow_{\sqcap}$, $\longrightarrow_{\sqcup}$, $\longrightarrow_{\forall}$, and $\longrightarrow_{\exists}^{\mathtt{K}}$. None of these rules are applicable on a closed ABox. The order of these rules affects only the performance of the system, with the exception of rule $\longrightarrow_{\exists}^{\mathtt{K}}$ that must be applied only when no other rule is applicable on the current set of ABoxes (to ensure termination). These rules roughly corresponds to deduction steps: they add assertions deduced from existing assertions.[1]

[1] To be more precise, each of them transforms a disjunction of ABoxes $\mathcal{D}$ into another disjunction of ABoxes $\mathcal{D}'$ such that given $\mathcal{T}_0$, $\mathcal{D}$ is satisfiable iff $\mathcal{D}'$ is satisfiable.

The rule $\longrightarrow_{\sqcap}$ is applicable on an ABox $\mathcal{A}$ if this latter contains an assertion of the form $(\mathtt{C}_1 \sqcap \ldots \sqcap \mathtt{C}_p)(\mathtt{a})$, and is such that at least one assertion $\mathtt{C}_k(\mathtt{a})$ $(1 \leq k \leq p)$ is not an element of $\mathcal{A}$. The application of this rule returns the ABox $\mathcal{A}'$ defined by

$$\mathcal{A}' = \mathcal{A} \cup \{\mathtt{C}_k(\mathtt{a}) \mid 1 \leq k \leq p\}$$

The rule $\longrightarrow_{\sqcup}$ is applicable on an ABox $\mathcal{A}$ if this latter contains an assertion of the form $(\mathtt{C}_1 \sqcup \ldots \sqcup \mathtt{C}_p)(\mathtt{a})$ but no assertion $\mathtt{C}_k(\mathtt{a})$ $(1 \leq k \leq p)$. The application of this rule returns the ABoxes $\mathcal{A}^1, \ldots, \mathcal{A}^p$ defined, for $1 \leq k \leq p$, by:

$$\mathcal{A}^k = \mathcal{A} \cup \{\mathtt{C}_k(\mathtt{a})\}$$

The rule $\longrightarrow_{\forall}$ is applicable on an ABox $\mathcal{A}$ if this latter contains two assertions, of respective forms $(\forall \mathtt{r}.\mathtt{C})(\mathtt{a})$ and $\mathtt{r}(\mathtt{a}, \mathtt{b})$ (with the same $\mathtt{r}$ and $\mathtt{a}$), and if $\mathcal{A}$ does not contain the assertion $\mathtt{C}(\mathtt{b})$. The application of this rule returns the ABox $\mathcal{A}'$ defined by

$$\mathcal{A}' = \mathcal{A} \cup \{\mathtt{C}(\mathtt{b})\}$$

The rule $\longrightarrow_{\exists}^{\mathtt{K}}$ is applicable on an ABox if

(i) $\mathcal{A}$ contains an assertion of the form $(\exists \mathtt{r}.\mathtt{C})(\mathtt{a})$;
(ii) $\mathcal{A}$ does not contain both an assertion of the form $\mathtt{r}(\mathtt{a}, \mathtt{b})$ and an assertion of the form $\mathtt{C}(\mathtt{b})$ (with the same $\mathtt{b}$, and with the same $\mathtt{C}$ and $\mathtt{a}$ as in previous condition);
(iii) There is no instance $\mathtt{c}$ such that $\{\mathtt{C} \mid \mathtt{C}(\mathtt{a}) \in \mathcal{A}\} \subseteq \{\mathtt{C} \mid \mathtt{C}(\mathtt{c}) \in \mathcal{A}\}$.[2]

If these conditions are applicable, let $\mathtt{b}$ be a new instance. The application of this rule returns the ABox $\mathcal{A}'$ defined by

$$\mathcal{A}' = \mathcal{A} \cup \{\mathtt{r}(\mathtt{a}, \mathtt{b}), \mathtt{C}(b)\} \cup \{\mathtt{K}(\mathtt{b})\}$$

Note that the TBox $\mathcal{T}_0 = \{\top \sqsubseteq \mathtt{K}\}$ is used here: since a new instance $\mathtt{b}$ is introduced, this instance must satisfy the TBox, which corresponds to the assertion $\mathtt{K}(\mathtt{b})$.

Remark 1. After the application of any of these rules on an ABox of $\mathcal{D}$, the resulting $\mathcal{D}'$ is equivalent to $\mathcal{D}$.

Example. Let us consider the example given at the end of section 2.1. Pretending that the source case represented by the instance σ can be applied to the target case represented by the instance θ amounts to identify these two instances, e.g., by substituting σ by θ. This leads to the ABox $\mathcal{A}_0 = \{\mathtt{Source}(\theta), \mathtt{Target}(\theta)\}$ (with $\mathtt{Source}$ and $\mathtt{Target}$ defined in (4)). The figure 1 represents this process. The entire tree represents the final set of ABoxes $\mathcal{D}_{\text{end}}$: each of the two branches represents a complete ABox $\mathcal{A} \in \mathcal{D}_{\text{end}}$. At the beginning of the process, the only nodes of this tree are $\mathtt{Source}(\theta)$, $\mathtt{Target}(\theta)$, and $\mathtt{K}(\theta)$: this corresponds to $\mathcal{D}_0 = \{\mathcal{A}_0^{\mathtt{K}}\}$. Then, the transformation rules are applied. Note that only the rule $\longrightarrow_{\sqcup}$ leads to branching. When a clash is detected in a branch (e.g. $\{\mathtt{Apple}(\mathtt{a}), (\neg \mathtt{Apple})(\mathtt{a})\}$) the branch represents a closed ABox (the clash is symbolised with $\square$). Note that the two final ABoxes are closed, meaning that $\{\mathtt{Source}(\theta), \mathtt{Target}(\theta)\}$ is not satisfiable: the source case needs to be adapted for being reused in the context of the target case.

[2] This third condition is called the *set-blocking* condition and is introduced to ensure the termination of the algorithm.

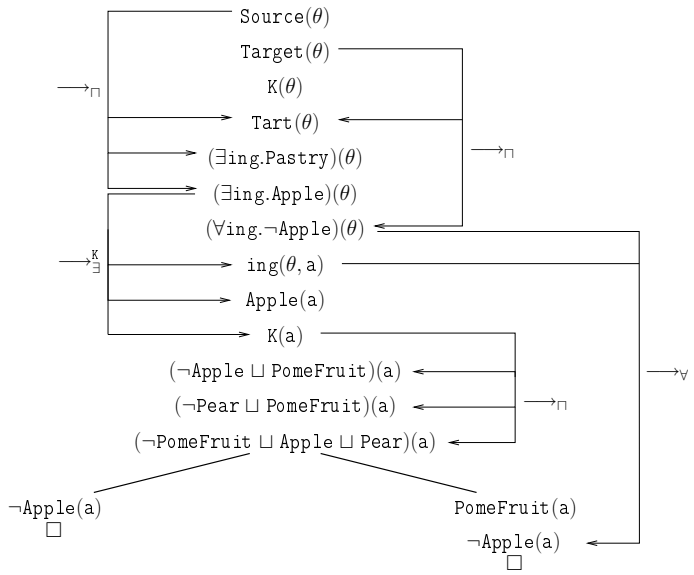

Fig. 1. Application of the tableau method proving that the ABox $\{\texttt{Source}(\theta), \texttt{Target}(\theta)\}$ is not satisfiable, given the TBox $\{\top \sqsubseteq \texttt{K}\}$ (for the sake of readability, the applications of the rules $\longrightarrow_{\sqcup}$ have not been represented; moreover, the order of application of rules has been chosen to make the example illustrative).

3 An Algorithm of Adaptation in $\mathcal{ALC}$

As seen in section 2.1, the reuse of the source case without adaptation may lead to a contradiction between $\texttt{Source}(\theta)$ and $\texttt{Target}(\theta)$. The adaptation algorithm presented in this section aims at solving this contradiction by weakening (generalising) $\texttt{Source}(\theta)$ so as to restore consistency, to apply to the target case θ what can be kept from Source.

3.1 Parameters and Result of the Algorithm

The parameters of the algorithm are DK, $\mathcal{A}^{\sigma}_{\texttt{srce}}$, $\mathcal{A}^{\theta}_{\texttt{tgt}}$, and cost. Its result is $\mathcal{D}$.

DK is a knowledge base in $\mathcal{ALC}$ representing the domain knowledge. In the running example, its ABox is empty, but in general, it may contain assertions.

The source and target cases are represented by two ABoxes that are satisfiable given DK: $\mathcal{A}^{\sigma}_{\texttt{srce}}$ and $\mathcal{A}^{\theta}_{\texttt{tgt}}$, respectively. More precisely, the source case is reified by an instance σ and $\mathcal{A}^{\sigma}_{\texttt{srce}}$ contains assertions about it. In the example above, $\mathcal{A}^{\sigma}_{\texttt{srce}}$ contains only one assertion, $\texttt{Source}(\sigma)$. Similarly, the target case is represented by an instance θ and $\mathcal{A}^{\theta}_{\texttt{tgt}}$ contains assertions about θ (only one assertion in the example: $\texttt{Target}(\theta)$).

The parameter cost is a function associating to a literal ℓ a numerical value $\texttt{cost}(\ell) > 0$, where a literal is either an atomic concept (positive literal) or a concept of the form $\neg\texttt{A}$ where A is atomic (negative literal). Intuitively, the greater $\texttt{cost}(\ell)$ is, the more difficult it is to give up the truth of an assertion $\ell(\texttt{a})$.

The algorithm returns $\mathcal{D}$, a set of ABoxes $\mathcal{A}$ solving the target case by adapting the source case: $\mathcal{A} \models \mathcal{A}^{\theta}_{\mathtt{tgt}}$ and $\mathcal{A}$ reuses "as much as possible" $\mathcal{A}^{\sigma}_{\mathtt{srce}}$. It may occur that $\mathcal{D}$ contains several ABoxes; in this situation, the knowledge of the system, in particular the `cost` function, is not complete enough to make a choice, thus it it up to the user to select an $\mathcal{A} \in \mathcal{D}$ (ultimately, by a random choice).

3.2 Steps of the Algorithm

The algorithm is composed of the following steps:

Preprocessing. Let $\mathcal{T}_{\mathtt{DK}}$ and $\mathcal{A}_{\mathtt{DK}}$ be the TBox and ABox of `DK`. Let `K` be a concept under NNF such that $\mathcal{T}_{\mathtt{DK}}$ is equivalent to $\{\top \sqsubseteq \mathtt{K}\}$. $\mathcal{A}_{\mathtt{DK}}$ is simply added to the ABoxes:

$$\mathcal{A}^{\sigma}_{\mathtt{srce}} \leftarrow \mathcal{A}^{\sigma}_{\mathtt{srce}} \cup \mathcal{A}_{\mathtt{DK}} \qquad\qquad \mathcal{A}^{\theta}_{\mathtt{tgt}} \leftarrow \mathcal{A}^{\theta}_{\mathtt{tgt}} \cup \mathcal{A}_{\mathtt{DK}}$$

Then, $\mathcal{A}^{\sigma}_{\mathtt{srce}}$ and $\mathcal{A}^{\theta}_{\mathtt{tgt}}$ are put under NNF.

Pretending that the source case solves the target problem. Reusing $\mathcal{A}^{\sigma}_{\mathtt{srce}}$ for the instance θ reifying the target case is done by assimilating the two instances σ and θ. This leads to the ABox $\mathcal{A}^{\theta}_{\mathtt{srce}}$, obtained by substituting σ by θ in $\mathcal{A}^{\sigma}_{\mathtt{srce}}$. Let $\mathcal{A}^{\theta}_{\mathtt{srce,tgt}} = \mathcal{A}^{\theta}_{\mathtt{srce}} \cup \mathcal{A}^{\theta}_{\mathtt{tgt}}$. If $\mathcal{A}^{\theta}_{\mathtt{srce,tgt}}$ is satisfiable given `DK`, then the straightforward reuse of the source case does not lead to any contradiction with the specification of the target case, so it just adds information about it. For example, let $\mathcal{A}^{\sigma}_{\mathtt{srce}} = \{\mathtt{Source}(\sigma)\}$ given by equation (3), let $\mathcal{A}^{\theta}_{\mathtt{tgt}} = \{\mathtt{Tart}(\theta), \mathtt{ing}(\theta, \mathtt{p}), \mathtt{FlakyPastry}(\mathtt{p})\}$ (i.e., "I want a tart with flaky pastry"), and the domain knowledge be $\mathtt{DK}' = \mathtt{DK} \cup \{\mathtt{FlakyPastry} \sqsubseteq \mathtt{Pastry}\}$, with `DK` defined in (2). With this example, it can be shown that $\mathcal{A}^{\theta}_{\mathtt{srce,tgt}}$ is satisfiable given $\mathtt{DK}'$ and it corresponds to an apple tart with flaky pastry.

In many situations, however, $\mathcal{A}^{\theta}_{\mathtt{srce,tgt}}$ is not satisfiable given `DK`. This holds for the running example. The principle of the adaptation algorithm consists in repairing $\mathcal{A}^{\theta}_{\mathtt{srce,tgt}}$. By "repairing" $\mathcal{A}^{\theta}_{\mathtt{srce,tgt}}$ we mean modifying it so as to make it complete and clash-free, and thus consistent. Removing clashes is not enough for that, the formulas from which they were were generated should be removed too. This motivates the introduction in section 3.2 of the AGraphs that extend ABoxes by keeping track of the application of rules. Moreover, to have a more fine-grained adaptation, $\mathcal{A}^{\theta}_{\mathtt{srce}}$ and $\mathcal{A}^{\theta}_{\mathtt{tgt}}$ are completed by tableau before being combined.

Applying the tableau method on $\mathcal{A}^{\theta}_{\mathtt{srce}}$ and on $\mathcal{A}^{\theta}_{\mathtt{tgt}}$, with memorisation of the transformation rule applications. In order to implement this step and the next ones, the notion of *assertional graph* (or *AGraph*) is introduced. An AGraph $\mathcal{G}$ is a simple graph whose set of nodes, $\mathit{Nodes}(\mathcal{G})$, is an ABox, and whose edges are labelled by transformation rules: if $(\alpha, \beta) \in \mathit{Edges}(\mathcal{G})$, the set of directed edges of $\mathcal{G}$, then $\lambda_{\mathcal{G}}(\alpha, \beta) = \varrho$ indicates that β has been obtained by applying ϱ on α and, possibly, on other assertions ($\lambda_{\mathcal{G}}$ is the labelling function of the graph $\mathcal{G}$).

The tableau method on AGraphs is based on the transformation rules $\Longrightarrow_{\sqcap}$, $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, and $\Longrightarrow_{\exists}^{\mathtt{K}}$. They are similar to the transformation rules on $\mathcal{ALC}$ ABoxes, with some differences.

The rule $\Longrightarrow_{\sqcap}$ is applicable on an AGraph $\mathcal{G}$ if

(i) $\mathcal{G}$ contains a node α of the form $(\mathtt{C}_1 \sqcap \ldots \sqcap \mathtt{C}_p)(\mathtt{a})$;
(ii) $\mathcal{G} \neq \mathcal{G}'$ (i.e., $\mathit{Nodes}(\mathcal{G}) \neq \mathit{Nodes}(\mathcal{G}')$ or $\mathit{Edges}(\mathcal{G}) \neq \mathit{Edges}(\mathcal{G}')$) with $\mathcal{G}'$ defined by

$$\begin{aligned}
\mathit{Nodes}(\mathcal{G}') &= \mathit{Nodes}(\mathcal{G}) \cup \{\mathtt{C}_k(\mathtt{a}) \mid 1 \leq k \leq p\} \\
\mathit{Edges}(\mathcal{G}') &= \mathit{Edges}(\mathcal{G}) \cup \{(\alpha, \mathtt{C}_k(\mathtt{a})) \mid 1 \leq k \leq p\} \\
\lambda_{\mathcal{G}'}(\alpha, \mathtt{C}_k(\mathtt{a})) &= \Longrightarrow_{\sqcap} \text{ for } 1 \leq k \leq p \\
\lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ for } e \in \mathit{Edges}(\mathcal{G})
\end{aligned}$$

Under these conditions, the application of the rule returns $\mathcal{G}'$.

The main difference between rule $\longrightarrow_{\sqcap}$ on ABoxes and rule $\Longrightarrow_{\sqcap}$ on AGraphs is that the latter may be applicable to $\alpha = (\mathtt{C}_1 \sqcap \ldots \sqcap \mathtt{C}_p)(\mathtt{a})$ even when $\mathtt{C}_k(\mathtt{a}) \in \mathit{Nodes}(\mathcal{G})$ for each k, $1 \leq k \leq p$. In this situation, $\mathit{Nodes}(\mathcal{G}') = \mathit{Nodes}(\mathcal{G})$ but $\mathit{Edges}(\mathcal{G}') \neq \mathit{Edges}(\mathcal{G})$: a new edge $(\alpha, \mathtt{C}_k)$ indicates here that $\alpha \models \mathtt{C}_k(\mathtt{a})$ and thus, if $\mathtt{C}_k(\mathtt{a})$ has to be removed, then α has also to be removed (see further, the repair step of the algorithm).

The rules $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, and $\Longrightarrow_{\exists}^{\mathtt{K}}$ are modified respectively from $\longrightarrow_{\sqcup}$, $\longrightarrow_{\forall}$, and $\longrightarrow_{\exists}^{\mathtt{K}}$ similarly. They are detailed in figure 2.

The tableau method presented in section 2.4 can be applied, given the TBox $\{\top \sqsubseteq \mathtt{K}\}$ and an ABox $\mathcal{A}_0$. The only difference is that AGraphs are manipulated instead of ABoxes, which involves that (1) an initial AGraph $\mathcal{G}_0$ has to be built from $\mathcal{A}_0$ (it is such that $\mathit{Nodes}(\mathcal{G}_0) = \mathcal{A}_0$ and $\mathit{Edges}(\mathcal{G}_0) = \emptyset$), (2) the rules $\Longrightarrow_{\cdot}$ are used instead of the rules $\longrightarrow_{\cdot}$, and (3) the result is a set of open and complete AGraphs (which is empty iff $\mathcal{G}_0$ is not satisfiable given $\{\top \sqsubseteq \mathtt{K}\}$).

Let $\{\mathcal{G}_i\}_{1 \leq i \leq m}$ and $\{\mathcal{H}_j\}_{1 \leq j \leq n}$ be the sets of open and complete AGraphs obtained by applying the tableau method respectively on $\mathcal{A}_0 = \mathcal{A}^{\theta}_{\mathtt{src}}$ and $\mathcal{A}^{\theta}_{\mathtt{tgt}}$. If $\mathcal{A}^{\theta}_{\mathtt{src}}$ and $\mathcal{A}^{\theta}_{\mathtt{tgt}}$ are satisfiable, then $m \neq 0$ and $n \neq 0$. If $m = 0$ or $n = 0$, the algorithm stops with value $\mathcal{D} = \{\mathcal{A}^{\theta}_{\mathtt{tgt}}\}$.

Generating explicit clashes from $\mathcal{G}_i$ and $\mathcal{H}_j$. A new kind of assertion, reifying the notion of clash, is considered: the *clash assertion* $\Box\pm\mathtt{A}(\mathtt{a})$ reifies the clash $\{\mathtt{A}(\mathtt{a}), (\neg\mathtt{A})(\mathtt{a})\}$. The rule $\Longrightarrow_{\Box}$ generates them. It is applicable on an AGraph $\mathcal{G}$ if

(i) $\mathcal{G}$ contains two nodes $\mathtt{A}(\mathtt{a})$ and $(\neg\mathtt{A})(\mathtt{a})$ (with the same $\mathtt{A}$ and the same $\mathtt{a}$);
(ii) $\mathcal{G} \neq \mathcal{G}'$ with $\mathcal{G}'$ defined by

$$\begin{aligned}
\mathit{Nodes}(\mathcal{G}') &= \mathit{Nodes}(\mathcal{G}) \cup \{\Box\pm\mathtt{A}(\mathtt{a})\} \\
\mathit{Edges}(\mathcal{G}') &= \mathit{Edges}(\mathcal{G}) \cup \{(\mathtt{A}(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a})), ((\neg\mathtt{A})(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a}))\} \\
\lambda_{\mathcal{G}'}(\mathtt{A}(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a})) &= \lambda_{\mathcal{G}'}((\neg\mathtt{A})(\mathtt{a}), \Box\pm\mathtt{A}(\mathtt{a})) = \Longrightarrow_{\Box} \\
\lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ for } e \in \mathit{Edges}(\mathcal{G})
\end{aligned}$$

A necessary condition for $\Longrightarrow_{\sqcup}$ to be applicable on an AGraph $\mathcal{G}$ is that $\mathcal{G}$ contains a node α of the form $(\mathtt{C}_1 \sqcup \ldots \sqcup \mathtt{C}_p)(\mathtt{a})$. If this is the case, then two situations can be considered:

(a) $\mathcal{G}$ contains no assertion $\mathtt{C}_k(\mathtt{a})$ $(1 \leq k \leq p)$. Under these conditions, the application of the rule returns the AGraphs $\mathcal{G}^1, \ldots, \mathcal{G}^p$ defined, for $1 \leq k \leq p$, by

$$\begin{aligned} \textit{Nodes}\,(\mathcal{G}^k) &= \textit{Nodes}\,(\mathcal{G}) \cup \{\mathtt{C}_k(\mathtt{a})\} \\ \textit{Edges}\,(\mathcal{G}^k) &= \textit{Edges}\,(\mathcal{G}) \cup \{(\alpha, \mathtt{C}_k(\mathtt{a}))\} \\ \lambda_{\mathcal{G}^k}(\alpha, \mathtt{C}_k(\mathtt{a})) &= \Longrightarrow_{\sqcup} \\ \lambda_{\mathcal{G}^k}(e) &= \lambda_{\mathcal{G}}(e) \text{ for } e \in \textit{Edges}\,(\mathcal{G}) \end{aligned}$$

(b) $\mathcal{G}$ contains one or several assertions $\beta_k = \mathtt{C}_k(\mathtt{a})$ such that $(\alpha, \beta_k) \notin \textit{Edges}\,(\mathcal{G})$. In this condition, $\Longrightarrow_{\sqcup}$ returns the AGraph $\mathcal{G}'$ obtained by adding to $\mathcal{G}$ these edges (α, β_k), with $\lambda_{\mathcal{G}'}(\alpha, \beta_k) = \Longrightarrow_{\sqcup}$.

The rule $\Longrightarrow_{\forall}$ is applicable on an AGraph $\mathcal{G}$ if

(i) $\mathcal{G}$ contains a node α_1 of the form $(\forall \mathtt{r}.\mathtt{C})(\mathtt{a})$ and a node α_2 of the form $\mathtt{r}(\mathtt{a}, \mathtt{b})$;

(ii) $\mathcal{G} \neq \mathcal{G}'$ with $\mathcal{G}'$ defined by

$$\begin{aligned} \textit{Nodes}\,(\mathcal{G}') &= \textit{Nodes}\,(\mathcal{G}) \cup \{\mathtt{C}(\mathtt{b})\} \\ \textit{Edges}\,(\mathcal{G}') &= \textit{Edges}\,(\mathcal{G}) \cup \{(\alpha_1, \mathtt{C}(\mathtt{b})), (\alpha_2, \mathtt{C}(\mathtt{b}))\} \\ \lambda_{\mathcal{G}'}(\alpha_1, \mathtt{C}(\mathtt{b})) &= \lambda_{\mathcal{G}'}(\alpha_2, \mathtt{C}(\mathtt{b})) = \Longrightarrow_{\forall} \\ \lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ for } e \in \textit{Edges}\,(\mathcal{G}) \end{aligned}$$

Under these conditions, the application of the rule returns $\mathcal{G}'$.

The rule $\Longrightarrow^{\mathtt{K}}_{\exists}$ is applicable on an AGraph $\mathcal{G}$ if

(i) $\mathcal{G}$ contains a node α of the form $(\exists \mathtt{r}.\mathtt{C})(\mathtt{a})$;

(ii) (a) Either $\mathcal{G}$ does not contain both $\mathtt{r}(\mathtt{a}, \mathtt{b})$ and $\mathtt{C}(\mathtt{b})$, for any instance $\mathtt{b}$;

(b) Or $\mathcal{G}$ contains two assertions $\beta_1 = \mathtt{r}(\mathtt{a}, \mathtt{b})$ and $\beta_2 = \mathtt{C}(\mathtt{b})$, such that $(\alpha, \beta_1) \notin \textit{Edges}\,(\mathcal{G})$ or $(\alpha, \beta_2) \notin \textit{Edges}\,(\mathcal{G})$;

(iii) There is no instance $\mathtt{c}$ such that $\{\mathtt{C} \mid \mathtt{C}(\mathtt{a}) \in \textit{Nodes}\,(\mathcal{G})\} \subseteq \{\mathtt{C} \mid \mathtt{C}(\mathtt{c}) \in \textit{Nodes}\,(\mathcal{G})\}$ (set-blocking condition, introduced for ensuring termination of the algorithm).

If condition (ii-a) holds, let $\mathtt{b}$ be a new instance. The application of the rule returns $\mathcal{G}'$ defined by

$$\begin{aligned} \textit{Nodes}\,(\mathcal{G}') &= \textit{Nodes}\,(\mathcal{G}) \cup \{\mathtt{r}(\mathtt{a}, \mathtt{b}), \mathtt{C}(\mathtt{b}), \mathtt{K}(\mathtt{b})\} \\ \textit{Edges}\,(\mathcal{G}') &= \textit{Edges}\,(\mathcal{G}) \cup \{(\alpha, \mathtt{r}(\mathtt{a}, \mathtt{b})), (\alpha, \mathtt{C}(\mathtt{b}))\} \\ \lambda_{\mathcal{G}'}(\alpha, \mathtt{r}(\mathtt{a}, \mathtt{b})) &= \lambda_{\mathcal{G}'}(\alpha, \mathtt{C}(\mathtt{b})) = \Longrightarrow^{\mathtt{K}}_{\exists} \\ \lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ for } e \in \textit{Edges}\,(\mathcal{G}) \end{aligned}$$

Under condition (ii-b), the application of the rule returns $\mathcal{G}'$ defined by

$$\begin{aligned} \textit{Nodes}\,(\mathcal{G}') &= \textit{Nodes}\,(\mathcal{G}) \\ \textit{Edges}\,(\mathcal{G}') &= \textit{Edges}\,(\mathcal{G}) \cup \{(\alpha, \beta_1), (\alpha, \beta_2)\} \\ \lambda_{\mathcal{G}'}(\alpha, \beta_1) &= \lambda_{\mathcal{G}'}(\alpha, \beta_2) = \Longrightarrow^{\mathtt{K}}_{\exists} \\ \lambda_{\mathcal{G}'}(e) &= \lambda_{\mathcal{G}}(e) \text{ for } e \in \textit{Edges}\,(\mathcal{G}) \end{aligned}$$

Fig. 2. The transformation rules $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, and $\Longrightarrow^{\mathtt{K}}_{\exists}$.

Under these conditions, the application of the rule returns $\mathcal{G}'$.

The next step of the algorithm is to apply the tableau method on each $\mathcal{G}_i \cup \mathcal{H}_j$, for each i and j, $1 \leq i \leq m$, $1 \leq j \leq n$, using the transformation rules $\Longrightarrow_{\sqcap}$, $\Longrightarrow_{\sqcup}$, $\Longrightarrow_{\forall}$, $\Longrightarrow_{\exists}^{\mathtt{K}}$, and $\Longrightarrow_{\Box}$. A difference with the tableau method presented above is that it was useless to apply rules on closed ABoxes (or closed AGraphs). Here, when a rule is applicable to an AGraph containing an assertion clash, it is applied, which may lead to several clashes in the same AGraph.

Remark 2. If an assertion clash $\Box\pm\mathtt{A}(\mathtt{a})$ is generated, then this clash is the consequence of assertions of both $\mathcal{G}_i$ and $\mathcal{H}_j$, otherwise, it would have been a clash generated at the previous step of the algorithm (since these two AGraphs are complete and open).

Repairing the assertion clashes. The previous step has produced a non-empty set $\mathcal{S}_{ij}$ of AGraphs, for each $\mathcal{G}_i \cup \mathcal{H}_j$. The repair step consists in repairing each of these AGraphs $\Gamma \in \mathcal{S}_{ij}$ and keeping only the ones that minimise the repair cost.[3] Let $\Gamma \in \mathcal{S}_{ij}$. If Γ contains no assertion clash, this involves that $\mathcal{G}_i \cup \mathcal{H}_j$ is satisfiable and so is $\mathcal{A}^{\theta}_{\mathtt{srce},\mathtt{tgt}}$: no adaptation is needed. If Γ contains $\delta \geq 1$ assertion clashes, then one of them is chosen and the repair according to this clash gives a set of repaired AGraphs Γ' containing $\delta - 1$ clashes. Then, the repair is resumed on Γ', until there is no more clash.[4] The cost of the global repair is the sum of the costs of each repair. In the following, it is shown how one clash of Γ is repaired.

The principle of the clash repair is to remove assertions of Γ in order to avoid this assertion clash to be re-generated by re-application of the rules. Therefore, the repair of all the assertion clashes must lead to satisfiable AGraphs (this is a consequence of the completeness of the tableau algorithm on $\mathcal{ALC}$). For this purpose, the following principle, expressed as an inference rule, is used:

$$\frac{\varphi \models \beta \qquad \beta \text{ has to be removed}}{\varphi \text{ has to be removed}} \tag{5}$$

where β is an assertion and φ is a minimal set of assertions such that $\varphi \models \beta$ (φ is to be understood as the conjunction of its formulas). Removing φ amounts to forget one of the assertions $\alpha \in \varphi$: when $\text{card}(\varphi) \geq 2$, there are several ways to remove φ, and thus, there may be several AGraphs Γ' obtained from Γ. The relation $\models$ linking φ and β is materialised by the edges of Γ. Therefore, on the basis of (5), the removal will be propagated by following these edges (α, β), from β to α.

Let $\beta = \Box\pm\mathtt{A}(\mathtt{a})$, the assertion clash of Γ to be removed. Let $\alpha^+ = \mathtt{A}(\mathtt{a})$ and $\alpha^- = \neg\mathtt{A}(\mathtt{a})$. At least one of α^+ and α^- has to be removed. $\mathcal{H}_j$ being an open and complete AGraph, either $\alpha^+ \notin \mathcal{H}_j$ or $\alpha^- \notin \mathcal{H}_j$. Three types of situation remain:

- If $\alpha^+ \in \mathcal{H}_j$ then α^+ cannot be removed: it is an assertion generated from $\mathcal{A}^{\theta}_{\mathtt{tgt}}$. Then, α^- has to be removed.

[3] In our prototypical implementation of this algorithm, this has been improved by pruning the repair tasks when their cost exceed the current minimum.

[4] Some additional nodes may have to be removed to ensure consistency of the repaired AGraph. They are determined by some technical analysis over the set-bockings ($\Longrightarrow_{\exists}^{\mathtt{K}}$, condition (iii)).

- If $\alpha^- \in \mathcal{H}_j$ then α^+ has to be removed.
- If $\alpha^+ \notin \mathcal{H}_j$ and $\alpha^- \notin \mathcal{H}_j$, then the choice of removal is based on the minimisation of the cost. If $\texttt{cost}(\texttt{A}) < \texttt{cost}(\neg\texttt{A})$ then α^+ has to be removed. If $\texttt{cost}(\texttt{A}) > \texttt{cost}(\neg\texttt{A})$ then α^- has to be removed. If $\texttt{cost}(\texttt{A}) = \texttt{cost}(\neg\texttt{A})$, then two AGraphs are generated: one by removing α^+, the other one, by removing α^-.

If an assertion β has to be removed, the propagation of the removal for an edge (α, β) such that $\lambda_{\mathcal{G}}(\alpha, \beta) \in \{\Longrightarrow_{\sqcap}, \Longrightarrow_{\sqcup}, \Longrightarrow^{\texttt{K}}_{\exists}\}$ consists in removing α (and propagating the removal from α).

Let β be an assertion to be removed that has been inferred by the rule $\Longrightarrow_{\forall}$. This means that there exist two assertions such that $\lambda_{\mathcal{G}}(\alpha_1, \beta) = \lambda_{\mathcal{G}'}(\alpha_2, \beta) = \Longrightarrow_{\forall}$. In this situation, two AGraphs are generated, one based on the removal of α_1, the other one, on the removal of α_2 (when α_1 or α_2 is in $\mathcal{H}_j$, only one AGraph is generated).

At the end of the repair process, a non empty set $\{\Gamma_k\}_{1 \leq k \leq p}$ of AGraphs without clashes has been built. Only the ones that are the result of a repair with a minimal cost are kept. Let $\mathcal{A}_k = \mathit{Nodes}(\Gamma_k)$. The result of the repair is $\mathcal{D} = \{\mathcal{A}_k\}_{1 \leq k \leq p}$.

Transforming the disjunction of ABoxes $\mathcal{D}$. If $\mathcal{A}, \mathcal{B} \in \mathcal{D}$ are such that $\mathcal{A} \models \mathcal{B}$, then the ABoxes disjunctions $\mathcal{D}$ and $\mathcal{D} \setminus \{\mathcal{A}\}$ are equivalent. This is used to simplify $\mathcal{D}$ by removing such $\mathcal{A}$.[5] After this simplifying test, each $\mathcal{A} \in \mathcal{D}$ is rewritten to remove the instances $\texttt{i}$ introduced during a tableau process. First, the $\texttt{i}$'s not related, neither directly, nor indirectly, to any non introduced instance by assertions $\texttt{r}(\texttt{a}, \texttt{b})$ are removed, meaning that the assertions with such $\texttt{i}$'s are removed (this may occur because of the repair step that may "disconnect" $\texttt{i}$ from non-introduced instances). Then, a "de-skolemisation" process is done by replacing the introduced instances $\texttt{i}$ by assertions of the form $(\exists \texttt{r}.\texttt{C})(\texttt{a})$. For instance, the set $\{\texttt{r}(\texttt{a}, \texttt{i}_1), \texttt{A}(\texttt{i}_1), \texttt{s}(\texttt{i}_1, \texttt{i}_2), \neg\texttt{B}(\texttt{i}_2)\}$ is replaced by $\{(\exists \texttt{r}.(\texttt{A} \sqcap \exists \texttt{s}.\neg\texttt{B}))(\texttt{a})\}$. The final value of $\mathcal{D}$ is returned by the algorithm.

Example. Consider the example given at the end of section 2.1. Giving all the steps of the algorithm is tedious, thus only the repairs will be considered.

Several AGraphs are generated and have to be repaired but they all share the same clash $\square\pm\texttt{Apple}(\texttt{a})$. Two repairs are possible and the resulting $\mathcal{D}$ depends only on the costs $\texttt{cost}(\texttt{Apple})$ and $\texttt{cost}(\neg\texttt{Apple})$.

If $\texttt{cost}(\texttt{Apple}) < \texttt{cost}(\neg\texttt{Apple})$, then $\mathcal{D} = \{\mathcal{A}\}$ with $\mathcal{A}$ equivalent to $(\texttt{Tart} \sqcap \exists \texttt{ing}.\texttt{Pear})(\theta)$. The proposed adaptation is a pear tart.

If $\texttt{cost}(\texttt{Apple}) \geq \texttt{cost}(\neg\texttt{Apple})$, then $\mathcal{D} = \{\mathcal{A}\}$, with $\mathcal{A}$ equivalent to $\mathcal{A}^{\theta}_{\texttt{tgt}}$. Nothing is learnt from the source case for the target case.

[5] In our tests, we have used necessary conditions of $\mathcal{A} \models \mathcal{B}$ based on set inclusions, with or without the renaming of one introduced instance. This has led to a dramatic reduction of the size of $\mathcal{D}$, which suggests that the algorithm presented above can be greatly improved, by pruning unnecessary ABox generation.

3.3 Properties of the Algorithm

The adaptation algorithm terminates. This can be proven using the termination of the tableau algorithm on ABoxes [2]. Repair removes at least one node from finite AGraphs at each step, thus it terminates too.

Every ABox $\mathcal{A} \in \mathcal{D}$ satisfies `Target` constraints: $\mathcal{A} \models \mathcal{A}^{\theta}_{\mathtt{tgt}}$.

Provided that $\mathcal{A}^{\theta}_{\mathtt{tgt}}$ is satisfiable, every $\mathcal{A} \in \mathcal{D}$ is satisfiable. In other words, unless the target case is in contradiction with the domain knowledge, the adaptation provides a consistent result. When $\mathcal{A}^{\theta}_{\mathtt{srce}}$ is not satisfiable, $\mathcal{D}$ is equivalent to $\{\mathcal{A}^{\theta}_{\mathtt{tgt}}\}$. This means that when a meaningless[6] $\mathcal{A}^{\theta}_{\mathtt{srce}}$ is given, $\mathcal{A}^{\theta}_{\mathtt{tgt}}$ is not altered.

If the source case is applicable under the target case constraints ($\mathcal{A}^{\theta}_{\mathtt{srce,tgt}} = \mathcal{A}^{\theta}_{\mathtt{srce}} \cup \mathcal{A}^{\theta}_{\mathtt{tgt}}$ is satisfiable) then $\mathcal{D}$ contains a sole ABox which is equivalent to $\mathcal{A}^{\theta}_{\mathtt{srce,tgt}}$: the source case is reused without modification to solve the target case.

The adaptation presented here can be considered as a generalisation and specialisation approach to adaptation. The ABoxes $\mathcal{A} \in \mathcal{D}$ are obtained by "generalising" $\mathcal{A}^{\theta}_{\mathtt{srce}}$ into $\mathcal{A}'$: some formulas of $\mathcal{A}^{\theta}_{\mathtt{srce}}$ are dropped for weaker consequences to obtain $\mathcal{A}'$ thus $\mathcal{A} \models \mathcal{A}'$, then $\mathcal{A}'$ is "specialised" into $\mathcal{A} = \mathcal{A}' \cup \mathcal{A}^{\theta}_{\mathtt{tgt}}$.

4 Discussion and related work

Beyond matching-based adaptation processes? There are two types of algorithms for the classical deductive inferences in DLs: the tableau algorithm presented above and the structural algorithms. The former is used for expressive DLs (i.e., for $\mathcal{ALC}$ and all the DLs extending $\mathcal{ALC}$). The latters are used for the other DLs (for which at least some of the deductive inferences are polynomial). A structural algorithm for the subsumption test $\mathtt{KB} \models \mathtt{C} \sqsubseteq \mathtt{D}$ consists, after a preprocessing step, in *matching* descriptors of `D` with descriptors of `C`. This matching procedure is rather close to the matching procedures used by most of the adaptation procedures, explicitly or not (if the cases have a fixed attribute-value structure, usually, the source and target cases are matched attribute by attribute, and the matching process does not need to be made explicit). Structural algorithms appear to be ill-suited for expressive DLs and tableau algorithms are used instead. The adaptation algorithm presented in this paper, based on tableau method principles, has no matching step (even if one can a posteriori match descriptors of source case and adapted target case). From those observations, we hypothesise that beyond a certain level of expressivity of the representation language, it becomes hardly possible to use matching techniques for an adaptation taking into account domain knowledge.

Other work on CBR and description logics. Despite the advantages of using DLs in CBR, as motivated in the introduction, there are rather few research on CBR and DLs. In [7], concepts of a DL are used as indexes for retrieving plans of a case-based planner, and adaptation is performed in another formalism. In [11], a non expressive DL is used for retrieval and for case base organisation. This work uses in particular the

[6] In a logical setting, an inconsistent knowledge base is equivalent to any other inconsistent knowledge base and thus, it is meaningless.

notion of *least common subsumer* (LCS) to reify similarity of the concepts representing the source and target cases: the LCS of concepts $\mathtt{C}$ and $\mathtt{D}$ is the more specific concept that is more general than both $\mathtt{C}$ and $\mathtt{D}$ and thus points out their common features. Therefore the LCS inference can be seen as a matching process (that might be used by some adaptation process). In an expressive DL, the LCS of $\mathtt{C}$ and $\mathtt{D}$ is $\mathtt{C} \sqcup \mathtt{D}$ (or an equivalent concept), which does not express anything about similar features of $\mathtt{C}$ and $\mathtt{D}$.

To our knowledge, the only attempts to define an adaptation process for DLs are [5] and [4]. [5] presents a modelling of the CBR life cycle using DLs. In particular, it presents a substitution approach to adaptation which consists in matching source and target case items by chains of roles (similar to chains of assertions $\mathtt{r}(\mathtt{a}_1, \mathtt{a}_2)$, $\mathtt{r}(\mathtt{a}_2, \mathtt{a}_3)$, etc.) in order to point out what substitutions can be done. [4] uses adaptation rules (reformulations) and multi-viewpoint representation for CBR, including a complex adaptation step. By contrast, the algorithm presented in this paper uses mainly the domain knowledge to perform adaptation: a direction of work will be to see how these approaches can be combined.

5 Conclusion and Future Work

This paper presents an algorithm for adaptation dedicated to case-based reasoning systems whose cases and domain knowledge are represented in the expressive DL $\mathcal{ALC}$. The first question raised by an adaptation problem is: "What has to be adapted?" The way this question is addressed by the algorithm consists in first pretending that the source case solves the target problem and then pointing out logical inconsistencies: these latters correspond to the parts of the source case to be modified in order to suit the target case. These principles are then applied to $\mathcal{ALC}$, for which logical inconsistencies are reified by the clashes generated by the tableau method. The second question raised by an adaptation problem is: "How will the source case be adapted?" The idea of the algorithm is to repair the inconsistencies by removing (temporarily) some knowledge from the source case, until the consistency is restored. This adaptation approach can be classified as a transformational one since it does not use explanations or justifications associated with the source case, as would a derivational (or generative) approach do.

Currently, only a basic prototype of this adaptation algorithm has been implemented, and it is not very efficient. A future work will aim at implementing it efficiently and in an extendable way, taking into account the future extensions presented below. This might be done by reusing available DL inference engines, provided their optimisation techniques do not interfere with the results of this adaptation procedure. It can be noted that the research on improving the tableau method for DLs has led to dramatic gains in term of computing time (see, in particular, [6]).

The second direction of work will be to extend the algorithm to other expressive DLs. In particular, we plan to extend it to $\mathcal{ALC}(\mathtt{D})$, where $\mathtt{D}$ is the concrete domain of real number tuples with linear constraint predicates. This means that cases may have numerical features (integer or real numbers) and domain knowledge may contain linear constraints on these features. This future work will also extend [3].

The algorithm of adaptation presented above can be considered as a generalisation and specialisation approach to adaptation (cf. section 3.3). By contrast, the algorithm

of [4] is a rule-based adaptation, a rule (a reformulation) specifying a relevant substitution to a given class of source case. A lead to integrate these two approaches is to use the adaptation rules during the repair process: instead of removing assertions leading to a clash, such a rule, when available, could be used to propose substitutes.

As written in the introduction, this algorithm follows work on adaptation based on belief revision, though it cannot be claimed that this algorithm, as such, implements a revision operator for $\mathcal{ALC}$ (e.g., it does not enable the revision of a TBox by an ABox). In [3], revision-based adaptation is generalised in merging-based case combination. Such a generalisation should be applicable to the algorithm defined in this paper: the ABox $\mathcal{A}^{\theta}_{\mathtt{srce}}$ is replaced by several ABoxes and the repairs are applied on these ABoxes. Defining precisely this algorithm and studying its properties is another future work.

Acknowledgements: The authors wish to thank the reviewers for their helpful comments and suggestions for future work.

References

1. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the Logic of Theory Change: partial meet functions for contraction and revision. Journal of Symbolic Logic 50, 510–530 (1985)
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook. Cambridge University Press, cambridge, UK (2003)
3. Cojan, J., Lieber, J.: Belief Merging-based Case Combination. In: David C. Wilson, Lorraine McGinty (eds.) 8th International Conference on Case-Based Reasoning - ICCBR 2009 Case-Based Reasoning Research and Development. Lecture Notes in Computer Science, vol. 5650, pp. 105–119. Springer Berlin, Seattle United States (07 2009)
4. d'Aquin, M., Lieber, J., Napoli, A.: Decentralized Case-Based Reasoning for the Semantic Web. In: Gil, Y., Motta, E. (eds.) Proceedings of the 4th International Semantic Web Conference (ISWC 2005). pp. 142–155. LNCS 3729, Springer (November 2005)
5. Gómez-Albarrán, M., González-Calero, P.A., Díaz-Agudo, B., Fernández-Conde, C.: Modelling the CBR Life Cycle Using Description Logics. In: Klaus-Dieter Althoff and Ralph Bergmann and L. Karl Branting (ed.) Proceedings of the 3rd International Conference on Case-Based Reasoning Research and Development (ICCBR-99). pp. 147–161. LNAI 1650, Springer, Berlin (1999)
6. Horrocks, I.: Optimising Tableaux Decision Procedures for Description Logics. Ph.D. thesis, University of Manchester (1997)
7. Koehler, J.: Planning from Second Principles. Artificial Intelligence 87, 145–186 (1996)
8. Kolodner, J.: Case-Based Reasoning. Morgan Kaufmann, Inc. (1993)
9. Lieber, J.: Application of the Revision Theory to Adaptation in Case-Based Reasoning: the Conservative Adaptation. In: Proceedings of the 7th International Conference on Case-Based Reasoning (ICCBR-07), pp. 239–253. LNAI 4626, Springer, Belfast (2007)
10. Riesbeck, C.K., Schank, R.C.: Inside Case-Based Reasoning. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey (1989)
11. Salotti, S., Ventos, V.: Study and Formalization of a Case-Based Reasoning System Using a Description Logic. In: Smyth, B., Cunningham, P. (eds.) Fourth European Workshop on Case-Based Reasoning, EWCBR-98. pp. 286–297. LNAI 1488, Springer (1998)