Analogical Transfer in RDFS, Application to Cocktail Name Adaptation

Nadia Kiani, Jean Lieber, Emmanuel Nauer, Jordan Schneider

Analogical Transfer in RDFS, Application to Cocktail Name Adaptation

Nadia Kiani[1], Jean Lieber[2], Emmanuel Nauer[2], and Jordan Schneider[1]

[1] Université de Lorraine,
SCA Master (Cognitive Sciences and their Applications), 54000 Nancy
Nadia.Kiani4@etu.univ-lorraine.fr, Jordan.Schneider1@etu.univ-lorraine.fr

[2] Université de Lorraine, LORIA — 54506 Vandœuvre-lès-Nancy, France
CNRS — 54506 Vandœuvre-lès-Nancy, France
Inria — 54602 Villers-lès-Nancy, France
Jean.Lieber@loria.fr, Emmanuel.Nauer@loria.fr

Abstract. This paper deals with analogical transfer in the framework of the representation language RDFS. The application of analogical transfer to case-based reasoning consists in reusing the problem-solution dependency to the context of the target problem; thus it is a general approach to adaptation. RDFS is a representation language that is a standard of the semantic Web; it is based on RDF, a graphical representation of data, completed by an entailment relation. A dependency is therefore represented as a graph representing complex links between a problem and a solution, and analogical transfer uses, in particular, RDFS entailment. This research work is applied (and inspired from) the issue of cocktail name adaptation: given a cocktail and a way this cocktail is adapted by changing its ingredient list, how can the cocktail name be modified?

Keywords: adaptation, analogical transfer, RDFS, cocktail name adaptation

1 Introduction

This paper presents an approach to analogical transfer in RDFS, with an application to cocktail name adaptation.

Adaptation is a research issue of case-based reasoning (CBR [11]) that has received some attention during the last years in the CBR community (see, e.g., [2, 7, 9]). In particular, this has been an issue for the competitors of the Computer Cooking Contests (CCCs). Such a CCC competitor is meant to answer cooking query problems, such as $\mathtt{Q}$ = "I want a dessert with pear but without orange", using a recipe book as a case base. TAAABLE is one of these competitors [4]. In TAAABLE, several adaptation issues have been tackled:

- Adaptation of ingredients stating, e.g., that the substitution $\mathtt{apple} \rightsquigarrow \mathtt{pear}$ (consisting in replacing apples with pears) applied to an apple pie recipe gives an answer to the query $\mathtt{Q}$;
- Adaptation of quantities (e.g., modifying the mass of granulated sugar in the recipe);

– Adaptation of the preparation: add, remove and/or re-order the preparation steps when needed.

These three adaptation issues have been addressed thanks to the principle of revision-based adaptation, i.e., adaptation based on belief revision [3] (the first one has also been addressed using other techniques).

In the 2014 edition of the CCC, the jury has suggested the issue of adapting recipe names. The CookingCAKE system [10] has addressed this challenge for the CCC-2015, using a few rules.[1] The applicative motivation of this paper is to address this issue for TAAABLE, with a recipe base restricted to cocktails.

Unlike the other adaptation issues addressed in TAAABLE, revision-based adaptation has not appeared as a useful guideline for adapting cocktail names. Indeed, revision-based adaptation can be understood as modifying the source case so that it becomes consistent with the target problem, given the domain knowledge, and, for many examples, the inconsistency of a cocktail name wrt a cocktail recipe was—at least—difficult to capture. By contrast, analogical transfer (AT) has appeared as a useful guideline for this issue. In works about AT, graph-based representations, such as semantic networks [12], are often used. RDFS is such a representation formalism and is the standard of the semantic Web that has been chosen for this work. Actually, it is also used by TUUURBINE, which is a generic retrieval engine that is used in TAAABLE [5]. If some adaptation strategies proposed in this paper are domain-dependent, it is hoped that other ones cover a broader range of applications, yet this work contributes to adaptation and, more specifically, to analogical transfer, in the representation framework of RDFS.

The paper starts with preliminaries recalling notions related to AT and to RDFS (§2), and describes its applicative issue (§3). Then, it follows the steps of the study. First, a collection of cocktail name adaptations has been gathered; section 4 describes this gathering and gives a few representative examples. These adaptations have been analyzed in details and section 5 exemplifies such an analysis. From that, several approaches to cocktail name adaptation are proposed that cover the majority of the examples (§6). This work is afterwards discussed (§7). Section 8 concludes and presents some directions for future work.

2 Preliminaries

2.1 Adaptation by analogical transfer

A *case* in a particular application is a chunk of experience that is frequently represented by a problem `pb` and a solution $\texttt{sol}(\texttt{pb})$ of `pb`, where the notions of problem and solution are application-dependent. A case from the case base is called a *source case*, denoted by a pair $(\texttt{srce}, \texttt{sol}(\texttt{srce}))$. The problem to be solved is called the *target*

[1] For instance, the adjective `"cheesy"` is added to the recipe name if the adapted recipe of a sandwich contains some cheese. This is not a published material, though. In the paper [10], the accent is put on other issues. The authors wish to thank Gilbert Müller and Ralph Bergmann for having given them some hints about the recipe adaptation in CookingCAKE.

problem, denoted by tgt. A classical way to perform the case-based inference consists in choosing a source case $(\texttt{srce}, \texttt{sol}(\texttt{srce}))$ judged similar to tgt (retrieval step) and in modifying $\texttt{sol}(\texttt{srce})$ into a solution $\texttt{sol}(\texttt{tgt})$ of tgt (adaptation step).

One approach to adaptation is analogical transfer (AT) that has been studied within the analogical reasoning community (see, e.g., [6, 14]). It can be seen as an approach to derivational analogy [1] often used in case-based planning: information about the way $\texttt{sol}(\texttt{srce})$ is derived from srce plays a critical role. Figure 1 presents notations related to AT.

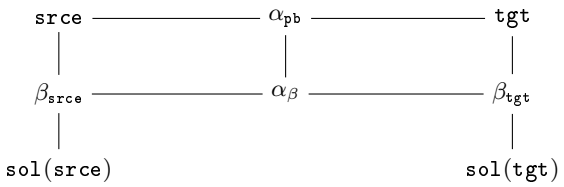


Fig. 1. Notations used to describe analogical transfer. tgt is the target problem. $(\texttt{srce}, \texttt{sol}(\texttt{srce}))$ is the retrieved case. $\beta_{\texttt{srce}}$ is a dependency between srce and $\texttt{sol}(\texttt{srce})$. $\alpha_{\texttt{pb}}$ is a matching from srce to tgt. From that, α_{β}, $\beta_{\texttt{tgt}}$ and $\texttt{sol}(\texttt{tgt})$ are inferred.

A *dependency* $\beta_{\texttt{pb}}$ between a problem pb and a solution $\texttt{sol}(\texttt{pb})$ of pb is constituted by pieces of information relating pb to $\texttt{sol}(\texttt{pb})$: it can be seen as a partial explanation of why $\texttt{sol}(\texttt{pb})$ solves pb. Given a source case $(\texttt{srce}, \texttt{sol}(\texttt{srce}))$, the dependency $\beta_{\texttt{srce}}$ can either be stored with the case, at case authoring time, or inferred.

The *matching* $\alpha_{\texttt{pb}}$ from a problem srce to a problem tgt is constituted by pieces of information about the differences and/or the similarities between the two problems. It is often inferred during retrieval time, the retrieved case being in general the one that best matches the target problem.

Given srce, $\texttt{sol}(\texttt{srce})$, $\beta_{\texttt{srce}}$, $\alpha_{\texttt{pb}}$ and tgt, the adaptation by AT can be described by the following steps:

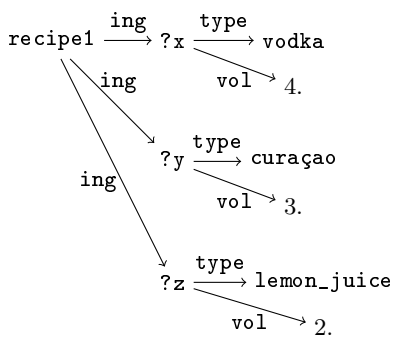
AT1 From $\beta_{\texttt{srce}}$ and $\alpha_{\texttt{pb}}$, infer a dependency $\beta_{\texttt{tgt}}$ between tgt and the (future) solution of tgt, $\texttt{sol}(\texttt{tgt})$. This inference consists in using the differences represented in $\alpha_{\texttt{pb}}$ to modify $\beta_{\texttt{srce}}$ into $\beta_{\texttt{tgt}}$. The matching between $\beta_{\texttt{srce}}$ and $\beta_{\texttt{tgt}}$ is denoted by α_{β}.

AT2 $\texttt{sol}(\texttt{srce})$ is modified into $\texttt{sol}(\texttt{tgt})$, using $\beta_{\texttt{tgt}}$ and α_{β}. The principle is to modify $\texttt{sol}(\texttt{srce})$ using α_{β} so that the result $\texttt{sol}(\texttt{tgt})$ respects the constraints given by $\beta_{\texttt{tgt}}$.

AT is an abstract approach for performing adaptation. To make it operational it is necessary to make some choices on the way the matchings and the dependencies are represented. For the latters, they are often represented in graph structures (e.g., in [6, 14]). Therefore, this justifies the use of RDFS formalism for representing and handling dependencies, as a way to implement AT.

2.2 RDFS

RDFS is a representation formalism based on RDF.



The "Blue Lagoon" recipe is identified by the resource `recipe1`. Its ingredients are 4 cl of vodka, 3 cl of curaçao, and 2 cl of lemon juice. The preparation is not represented. `ing` relates a recipe to one of its ingredients. `type` (abbreviation of `rdf:type`) is an RDF property relating a class to its instance (for example, the triple ⟨`?x type vodka`⟩ means that `?x` is an instance of `vodka`. The variables are existentially quantified (there exist `?x`, `?y` and `?z` such that...). The property `vol` relates an ingredient to its volume in centilitres.

Fig. 2. An RDF representation of the "Blue Lagoon" recipe.

RDF (*Resource Description Framework*[2]) is a language that can be used to encode assertions using triples, e.g. "Romeo loves Juliet and knows someone whose age is 40" can be encoded by:

⟨`romeo loves juliet`⟩ ⟨`romeo knows ?x`⟩ ⟨`?x age 40`⟩

In a *triple* $\langle s\ p\ o\rangle$, s (the subject) is a *resource*, p (the predicate) is a *property*, and o (the object) is either a resource or a *literal*. A resource is either a constant or a variable (generally called identified resource and blank node, respectively). By naming convention, variables start with the symbol `?` whereas constants do not. So, `juliet` is a constant and `?x` is a variable. A property is a particular type of resource, intended to represent a binary relation. A set of simple types (including `integer`, `float` and `string`) is fixed and a literal is a value of one of these types. For the sake of simplicity, triples $\langle s\ p\ o\rangle$ where s is a literal are also accepted in this paper, though this is not compliant with the RDFS standard: this will make some explanations simpler, avoiding some useless technicalities. An *RDF base* is a set of triples. An *RDF graph* is the graphical representation of an RDF base by a graph whose nodes are resources and literals, and whose edges are labeled by properties. For example, the RDF graph of figure 2 represents a cocktail recipe. Given an RDF base $\mathcal{B}$, the set of nodes of the corresponding RDF graph is denoted by $\texttt{Nodes}(\mathcal{B})$. Given $n_1, n_2 \in \texttt{Nodes}(\mathcal{B})$, n_1 and n_2 are *connected* in $\mathcal{B}$ if there exists a non-directed path relating them in the graph corresponding to $\mathcal{B}$.

[2] https://www.w3.org/RDF/

RDFS (*RDF Schema*[3]) is a representation formalism whose syntax is RDF and semantics is defined by a set of inference rules. Only a few rules are used in this paper:

$$\frac{\langle \texttt{a type C}\rangle \quad \langle \texttt{C subc D}\rangle}{\langle \texttt{a type D}\rangle} r_1 \qquad \frac{\langle \texttt{a p b}\rangle \quad \langle \texttt{p subp q}\rangle}{\langle \texttt{a q b}\rangle} r_2$$

$$\frac{\langle \texttt{A subc B}\rangle \quad \langle \texttt{B subc C}\rangle}{\langle \texttt{A subc C}\rangle} r_3 \qquad \frac{\langle p \texttt{ subc } q\rangle \quad \langle q \texttt{ subc } r\rangle}{\langle p \texttt{ subc } r\rangle} r_4$$

`type`, `subc` and `subp` are abbreviations for `rdf:type`, `rdfs:subClassOf` and `rdfs:subPropertyOf`. `type` is the membership relation between an instance and a class. `subc` (resp., `subp`) is the relation between a class and a superclass (resp., a property and a superproperty). r_1 means that if `a` is an instance of a class it is also an instance of its superclasses. r_2 means that if `a` and `b` are related by a property, they are also related by any of its superproperties. r_3 and r_4 state that `subc` and `subp` are transitive. For example, the following inference can be drawn:

$$\left\{ \begin{array}{c} \langle \texttt{?x type vodka}\rangle, \\ \langle \texttt{vodka subc alcoholicBeverage}\rangle \end{array} \right\} \vdash \langle \texttt{?x type alcoholicBeverage}\rangle$$

RDFS does not include negation, thus only positive facts can be entailed. However, an inference with closed world assumption (CWA) can be drawn, stating that if $\mathcal{B} \nvdash t$ then t is considered to be false (given the RDFS base $\mathcal{B}$), denoted by $\mathcal{B} \vdash_{\text{cwa}} \neg t$.

SPARQL (*SPARQL Protocol and RDF Query Language*[4]) enables to write queries to RDF or RDFS bases. If a SPARQL engine uses RDFS entailment, this means that the query is done on the RDF base completed by RDFS entailment. For example, the following SPARQL query addressed to a base describing recipes such as the one of figure 2 returns the set of recipes `?r` containing some alcohol:[5]

$$\texttt{Q}_{\texttt{alcohol}} = \text{SELECT } \texttt{?r} \text{ WHERE } \{\texttt{?r ing ?a . ?a type alcoholicBeverage}\} \quad (1)$$

Given a SPARQL query `Q` and an RDFS base $\mathcal{B}$, the result of the execution of `Q` on $\mathcal{B}$ is denoted by $\texttt{exec}(\texttt{Q}, \mathcal{B})$.

3 The cocktail name adaptation issue

In this application, a problem `pb` is a representation of a cocktail recipe by an RDFS graph. For the first version of this application, only ingredient types are considered, neither the quantities, nor the preparation steps. Therefore, a problem is an RDFS base $\texttt{pb} = \bigcup_{k=1}^{n} \{\langle id \texttt{ ing ?v}_k\rangle, \langle \texttt{?v}_k \texttt{ type } f_k\rangle\}$ where id is a constant (a resource identifying the recipe), $\texttt{?v}_1, \ldots, \texttt{?v}_n$ are n variables, and $f_1, \ldots, f_n$ are food classes.

[3] https://www.w3.org/2001/sw/wiki/RDFS

[4] https://www.w3.org/2001/sw/wiki/SPARQL

[5] The CWA is assumed: if it cannot be entailed that a recipe contains some alcohol, then it is concluded that it does not.

A solution $\texttt{sol}(\texttt{pb})$ of pb is a literal of type string that gives a name to pb. It is assumed to be in lower case for the sake of simplicity, e.g., $\texttt{sol}(\texttt{pb}) = \texttt{"blue lagoon"}$ solves the problem pb represented in figure 2. The following operations on strings are used: concatenation (denoted by $+$, e.g., $\texttt{"ab"} + \texttt{"cd"} = \texttt{"abcd"}$), substring checking (denoted by subStringOf, e.g., $\texttt{subStringOf}(\texttt{"bc"}, \texttt{"abcd"}) = \texttt{true}$), and string replacement (e.g., $\texttt{replace}(\texttt{"ab"}, \texttt{"cd"}, \texttt{"bababa"}) = \texttt{"bcdcda"}$).

A dependency $\beta_{\texttt{pb}}$ between pb and $\texttt{sol}(\texttt{pb})$ is an RDFS base. Usually, at least one food class f_k of pb and the literal $\texttt{sol}(\texttt{pb})$ occur in $\beta_{\texttt{pb}}$: when it is not the case, $\beta_{\texttt{pb}}$ does not relate pb to $\texttt{sol}(\texttt{pb})$ (which is possible, e.g., when $\beta_{\texttt{pb}} = \emptyset$, i.e., there is no known dependency between pb and $\texttt{sol}(\texttt{pb})$). For each case $(\texttt{srce}, \texttt{sol}(\texttt{srce}))$, $\beta_{\texttt{srce}}$ is assumed to be given.

A matching $\alpha_{\texttt{pb}}$ from srce to tgt is either simple or complex. A simple matching has the form $f \rightsquigarrow g$ where f is a food class of srce and g is a food class of tgt; it represents the substitution of f by g. The removal of a food class f will be denoted by $f \rightsquigarrow \emptyset$. A complex matching is a composition $\alpha_{\texttt{pb}} = \alpha^q_{\texttt{pb}} \circ \alpha^{q-1}_{\texttt{pb}} \circ \ldots \circ \alpha^1_{\texttt{pb}}$ of simple matchings. $\alpha_{\texttt{pb}}$ is built during the adaptation of ingredients process of TAAABLE.

The matching α_β from $\beta_{\texttt{srce}}$ to $\beta_{\texttt{tgt}}$ is built during the cocktail name adaptation. It consists of a set of ordered pairs (d, d') where d is a descriptor of $\beta_{\texttt{srce}}$ and d' is a descriptor of $\beta_{\texttt{tgt}}$, a descriptor being either a resource (that can be a property) or a literal.

Finally, the domain knowledge is represented by an RDFS base DK.

4 Collecting examples of cocktail name variations

The first step of this work has been to collect 20 examples of cocktail name adaptations, with an attempt to have diverse types of adaptation. 10 of them have been taken from variants of classical cocktails.[6] The other 10 have been imagined for this study.

Here is a selection of examples, knowing that for some of them, only the relevant part of the information has been given (hence the "etc."):

ex. 1 srce contains the ingredient classes vodka, curaçao and lemonJuice, $\texttt{sol}(\texttt{srce}) = \texttt{"blue lagoon"}$, $\alpha_{\texttt{pb}} = \texttt{curaçao} \rightsquigarrow \texttt{appleJuice}$, $\texttt{sol}(\texttt{tgt}) = \texttt{"yellow lagoon"}$. $\texttt{sol}(\texttt{srce})$ depends on the color of curaçao, which is substituted by a yellow beverage.

ex. 2 srce contains rhum, mintLeave, lime, brownSugar and ice, $\texttt{sol}(\texttt{srce}) = \texttt{"mojito"}$, $\alpha_{\texttt{pb}} = \texttt{rhum} \rightsquigarrow \emptyset$, $\texttt{sol}(\texttt{tgt}) = \texttt{"virgin mojito"}$. For some reason (that we do not wish to justify), transforming a recipe with alcohol into a recipe without alcohol makes it virgin.

ex. 3 srce contains scotchWhisky and amaretto, $\texttt{sol}(\texttt{srce}) = \texttt{"godfather"}$, $\alpha_{\texttt{pb}} = \texttt{scotchWhisky} \rightsquigarrow \texttt{irishWhisky}$, $\texttt{sol}(\texttt{tgt}) = \texttt{"the new godfather"}$. No explanation of the name is known, so a default rule proposes a variant name (it could have been $\texttt{"godfather 2"}$).

[6] In particular, http://www.1001cocktails.com/ has proven to be useful, since it contains descriptions of cocktails with some named variants.

ex. 4 srce contains irishWhisky, coffee, etc., $\texttt{sol}(\texttt{srce}) = \texttt{"irish coffee"}$, $\alpha_{\texttt{pb}} = \texttt{irishWhisky} \rightsquigarrow \texttt{tequila}$, $\texttt{sol}(\texttt{tgt}) = \texttt{"mexican coffee"}$. An Irish ingredient is replaced with a Mexican one.

ex. 5 $(\texttt{srce}, \texttt{sol}(\texttt{srce}))$ is the case of example 1, $\alpha_{\texttt{pb}} = \texttt{curaçao} \rightsquigarrow \texttt{indianTonic}$, two solutions are proposed: $\texttt{sol}(\texttt{tgt}) = \texttt{"bitter lagoon"}$ and $\texttt{sol}(\texttt{tgt}) = \texttt{"sparkling lagoon"}$. Indeed, blue is an organoleptic property of curaçao, whereas bitter and sparkling are organoleptic properties of Indian tonic.

5 From blue lagoon to yellow lagoon: analysis of an example

This section models the adaptation example 1 following the two steps of AT introduced in section 2.1.

AT 1. A partial explanation of the name $\texttt{sol}(\texttt{srce}) = \texttt{"blue lagoon"}$ is that the color of curaçao is blue (this is partial, since it does not explain the term "lagoon"), which can be modeled by

$$\beta_{\texttt{srce}} = \{\langle \texttt{curaçao color blue}\rangle, \langle \texttt{blue inEnglish "blue"}\rangle, \langle \texttt{"blue" subStringOf "blue lagoon"}\rangle\}$$

Since $\alpha_{\texttt{pb}} = \texttt{curaçao} \rightsquigarrow \texttt{appleJuice}$, in order to build $\beta_{\texttt{tgt}}$, the idea is to apply $\alpha_{\texttt{pb}}$ on $\beta_{\texttt{srce}}$ and then to make some modifications on the resources and literals to make it consistent with DK. This consistency test must be considered wrt CWA because there is no way to have $\langle \texttt{appleJuice color blue}\rangle$ inconsistent with DK in the classical semantics. It is assumed that $\texttt{DK} \vdash_{\text{cwa}} \neg\langle \texttt{appleJuice color blue}\rangle$, thus the mere substitution $\alpha_{\texttt{pb}}$ on $\beta_{\texttt{srce}}$ gives an inconsistent result wrt DK under CWA. So, the idea is to relax this triple. One way to do it is to replace blue with a variable ?x. More generally, the strategy consists in replacing the descriptors of $\beta_{\texttt{srce}}$ by variables, with the exception of the predicates (that are higher order resources) and of the descriptors occurring in tgt. The variable that replaces $\texttt{sol}(\texttt{srce})$ is ?solTgt: solving tgt consists in giving a value $\texttt{sol}(\texttt{tgt})$ to this variable. This gives the following dependency (obtained by applying $\alpha_{\texttt{pb}}$ and turning some constants into variables):

$$\beta_{\texttt{gen}} = \{\langle \texttt{appleJuice color ?x}\rangle, \langle \texttt{?x inEnglish ?y}\rangle, \langle \texttt{?y subStringOf ?solTgt}\rangle\}$$

$\beta_{\texttt{gen}}$ is so-called, since it generalizes $\alpha_{\texttt{pb}}(\beta_{\texttt{srce}})$ (in the sense $\alpha_{\texttt{pb}}(\beta_{\texttt{srce}}) \vdash \beta_{\texttt{gen}}$), where $\alpha_{\texttt{pb}}(\beta_{\texttt{srce}})$ is the result of applying the substitution $\alpha_{\texttt{pb}}$ on $\beta_{\texttt{srce}}$.

Now, in order to get $\beta_{\texttt{tgt}}$, the idea is to unify the variables ?x and ?y with some constants, using the domain knowledge. Therefore DK is interrogated with the following SPARQL query:

SELECT ?x ?y WHERE {appleJuice color ?x . ?x inEnglish ?y}

Assuming the only result is the pair $\{\texttt{?x} \leftarrow \texttt{yellow}, \texttt{?y} \leftarrow \texttt{"yellow"}\}$, it comes:

$$\beta_{\texttt{tgt}} = \{\langle \texttt{appleJuice color yellow}\rangle, \langle \texttt{yellow inEnglish "yellow"}\rangle, \langle \texttt{"yellow" subStringOf ?solTgt}\rangle\}$$
$$\text{and } \alpha_\beta = \{(\texttt{curaçao}, \texttt{appleJuice}), (\texttt{blue}, \texttt{yellow}), (\texttt{"blue"}, \texttt{"yellow"})\}$$

AT 2. Therefore, $\beta_{\texttt{tgt}}$ involves that $\texttt{sol}(\texttt{tgt})$ has to respect the following constraint:

$$\texttt{sol}(\texttt{tgt}) \in \{s : \text{string} \mid \texttt{"yellow"} \text{ is a substring of } s\} \tag{2}$$

Now, $\texttt{sol}(\texttt{srce})$ must be modified using α_β into $\texttt{sol}(\texttt{tgt})$ that respects (2). Here, a domain-dependent choice has to be made: it concerns the way the solution space is structured, i.e., how can modifications be applied on solutions. It is assumed that in this application, the only modification operation is based on the `replace` operation on the set of strings (which is the solution space). Hence, since $(\texttt{"blue"}, \texttt{"yellow"}) \in \alpha_\beta$, the following cocktail name that is consistent with (2) is proposed:

$$\texttt{sol}(\texttt{tgt}) = \texttt{replace}(\texttt{"blue"}, \texttt{"yellow"}, \texttt{sol}(\texttt{srce})) = \texttt{"yellow lagoon"}$$

6 Cocktail name adaptation strategies

An adaptation strategy is a function with the following signature:

input `srce`, $\texttt{sol}(\texttt{srce})$, `tgt`, $\beta_{\texttt{srce}}$, $\alpha_{\texttt{pb}}$, `DK`;
output a set of strings, each of them being a proposed solution $\texttt{sol}(\texttt{tgt})$ for `tgt`.

When the output is empty, this means that the strategy has failed. Each element of the output have to be different from $\texttt{sol}(\texttt{srce})$.

The two first strategies presented below (§6.1 and §6.2) are application-dependent, whereas the last ones should be adaptable to other applications. Strategies presented in sections 6.3 and 6.4 are designed for simple matchings whereas the strategy of section 6.5 combines strategies for dealing with complex matchings.

```
function adaptNameWhenRemovingAlcohol(...)
begin
    if srce contains some alcohol and tgt does not then
        sol(tgt) ← "virgin " + sol(srce)
        return {sol(tgt)}
    else
        return ∅
    end
end
```

(a) Adaptation strategy of §6.1.

```
function adaptDefault(...)
begin
    sol(tgt) ← "the new "
               + sol(srce)
    return {sol(tgt)}
end
```

(b) Adaptation strategy of §6.2.

Fig. 3. Two basic cocktail name adaptation strategies.

6.1 Strategy "Alcohol abuse is dangerous for health"

A simple strategy consists in generalizing the example 2. It is presented by the algorithm of figure 3(a). Note that the condition of the test can be performed by executing the SPARQL query $\mathtt{Q}_{\mathtt{alcohol}}$ (cf. equation (1)) twice:

- "$\mathtt{srce}$ contains some alcohol" is encoded by $\mathtt{exec}(\mathtt{Q}_{\mathtt{alcohol}}, \mathtt{DK} \cup \mathtt{srce}) \neq \emptyset$ and
- "$\mathtt{tgt}$ contains no alcohol" is encoded by $\mathtt{exec}(\mathtt{Q}_{\mathtt{alcohol}}, \mathtt{DK} \cup \mathtt{tgt}) = \emptyset$.

6.2 Default strategy

The default strategy is applied when all other strategies fail. It is presented by the algorithm of figure 3(b). Example 3 is an application of this strategy.

6.3 Strategy "Turn constants into variables"

This strategy has been generalized from the analysis of the example 1 that is described in section 5. Its algorithm is presented in figure 4. The two first tests of the algorithm define conditions under which the strategy applies ($\emptyset$ is returned otherwise): $\alpha_{\mathtt{pb}}$ is a substitution of ingredient $f \rightsquigarrow g$ and $\beta_{\mathtt{srce}}$ relates f to $\mathtt{sol}(\mathtt{srce})$.

Then the AT1 step of analogical transfer is implemented. First, $\widehat{\alpha_\beta}$ is computed: it corresponds to pairs (d, d') where d is a descriptor of $\beta_{\mathtt{srce}}$ and d' is either a variable or a value (constant or literal). New variables d' are generated that correspond to nodes of $\beta_{\mathtt{srce}}$ that are connected to the substituted ingredient class f. Second, $\beta_{\mathtt{gen}}$ is computed by replacing in $\beta_{\mathtt{srce}}$ d by d' for each $(d, d') \in \widehat{\alpha_\beta}$. Third, $\widehat{\alpha_\beta}$ and $\beta_{\mathtt{gen}}$ are instantiated by α_β and $\beta_{\mathtt{tgt}}$; since there may be several instantiations, pairs $(\alpha_\beta, \beta_{\mathtt{tgt}})$ are generated. To find these instantiations of variables, the domain knowledge is queried: a SPARQL query is built that enables to find variable instantiations respecting the constraints of $\beta_{\mathtt{tgt}}$.

Then, for each pair $(\alpha_\beta, \beta_{\mathtt{tgt}})$ so generated, the AT2 step is applied. It consists mainly in applying the function `modifyUnderMappingAndConstraints` that is described by the algorithm of figure 5: substrings of $\mathtt{sol}(\mathtt{srce})$ occurring in $\beta_{\mathtt{srce}}$ are replaced by the corresponding strings in $\beta_{\mathtt{tgt}}$. In theory, this algorithm is underspecified, since the order of the replacements matters for the result. In practice, however, in all the examples we have met, this has had no influence.

It is worth noticing that the only part of this strategy that is domain-dependent lies in the function `modifyUnderMappingAndConstraints` and that this latter depends essentially on information on how to "travel" in the solution space, that is, for the application presented here, the substring checking and the `replace` function.

Another illustration of this algorithm is given below. Consider the example 4, with the following dependency:

$$\begin{aligned}\beta_{\mathtt{srce}} = \{&\langle \mathtt{irishWhisky\ origin\ ireland} \rangle,\\ &\langle \mathtt{ireland\ englishAdjective\ "irish"} \rangle,\\ &\langle \mathtt{"irish"\ subStringOf\ "irish\ coffee"} \rangle,\\ &\langle \mathtt{coffee\ nameInEnglish\ "coffee"} \rangle,\\ &\langle \mathtt{"coffee"\ subStringOf\ "irish\ coffee"} \rangle\}\end{aligned}$$

function $\texttt{turnConstantsIntoVariables}(\texttt{srce}, \texttt{sol}(\texttt{srce}), \texttt{tgt}, \beta_{\texttt{srce}}, \alpha_{\texttt{pb}}, \texttt{DK})$
begin
  ▷ **testing whether the strategy applies**
  if $\alpha_{\texttt{pb}}$ *is not a simple matching* **then**
    return $\emptyset$ ▷ adaptation strategy failure
  end
  Let f, g be resources such that $\alpha_{\texttt{pb}} = f \rightsquigarrow g$ (f is necessarily a food class of $\texttt{srce}$).
  if f *is not connected to* $\texttt{sol}(\texttt{srce})$ *in* $\beta_{\texttt{srce}}$ **then**
    return $\emptyset$
  end
  ▷ **computing** $\widehat{\alpha_\beta}$
  $\widehat{\alpha_\beta} \leftarrow \{(f, g), (\texttt{sol}(\texttt{srce}), \texttt{?solTgt})\}$
  Add to $\widehat{\alpha_\beta}$ every (a, a) such that $a \in \texttt{Nodes}(\beta_{\texttt{srce}})$ and a is not connected to f in $\beta_{\texttt{srce}}$.
  for $n \in \texttt{Nodes}(\texttt{srce})$ **do**
    if *there is no image of* n *by* $\widehat{\alpha_\beta}$ **then**
      $n' \leftarrow$ new variable
      $\widehat{\alpha_\beta} \leftarrow \widehat{\alpha_\beta} \cup \{(n, n')\}$
    end
  end
  ▷ **computing** $\beta_{\texttt{gen}}$
  $\beta_{\texttt{gen}} \leftarrow \emptyset$
  for $\langle s\ \ p\ \ o\rangle \in \beta_{\texttt{srce}}$ **do**
    $s' \leftarrow \widehat{\alpha_\beta}(s)$ ▷ i.e., s' is such that $(s, s') \in \widehat{\alpha_\beta}$
    $o' \leftarrow \widehat{\alpha_\beta}(o)$
    $\beta_{\texttt{gen}} \leftarrow \beta_{\texttt{gen}} \cup \{\langle s'\ \ p\ \ o'\rangle\}$
  end
  ▷ **computing the set** $P_{\alpha_\beta \beta_{\texttt{tgt}}}$ **of ordered pairs** $(\alpha_\beta, \beta_{\texttt{tgt}})$
  $P_{\alpha_\beta \beta_{\texttt{tgt}}} \leftarrow \emptyset$
  Let $\texttt{Q}$ be the SPARQL query such that the selected variables of $\texttt{Q}$ are the variables occurring in $\widehat{\alpha_\beta}$ different from $\texttt{?solTgt}$ and the body of $\texttt{Q}$ (i.e., what follows WHERE) is constituted by triples of $\beta_{\texttt{gen}}$, except the ones with $\texttt{?solTgt}$
  $\texttt{R} \leftarrow \texttt{exec}(\texttt{Q}, \texttt{DK})$ ▷ $\texttt{R}$ is a set of assignments of the variables
  for $A \in \texttt{R}$ **do**
    $\alpha_\beta \leftarrow$ result of applying the assignment A on $\widehat{\alpha_\beta}$
    $\beta_{\texttt{tgt}} \leftarrow$ result of applying the assignment A on $\beta_{\texttt{gen}}$
    $P_{\alpha_\beta \beta_{\texttt{tgt}}} \leftarrow P_{\alpha_\beta \beta_{\texttt{tgt}}} \cup \{(\alpha_\beta, \beta_{\texttt{tgt}})\}$
  end
  ▷ **computing the set** $\texttt{SOLs}(\texttt{tgt})$ **of candidate values for** $\texttt{sol}(\texttt{tgt})$
  $\texttt{SOLs}(\texttt{tgt}) \leftarrow \emptyset$
  for $(\alpha_\beta, \beta_{\texttt{tgt}}) \in P_{\alpha_\beta \beta_{\texttt{tgt}}}$ **do**
    $\texttt{sol}(\texttt{tgt}) \leftarrow \texttt{modifyUnderMappingAndConstraints}(\texttt{sol}(\texttt{srce}), \beta_{\texttt{srce}}, \alpha_\beta, \beta_{\texttt{tgt}})$
    if $\texttt{sol}(\texttt{tgt}) \neq \texttt{sol}(\texttt{srce})$ **then**
      $\texttt{SOLs}(\texttt{tgt}) \leftarrow \texttt{SOLs}(\texttt{tgt}) \cup \{\texttt{sol}(\texttt{tgt})\}$
    end
  end
  return $\texttt{SOLs}(\texttt{tgt})$
end

Fig. 4. The turn constants into variables strategy.

function modifyUnderMappingAndConstraints($\mathtt{sol}(\mathtt{srce}), \beta_{\mathtt{srce}}, \alpha_\beta, \beta_{\mathtt{tgt}}$)
begin
  $\mathtt{sol}(\mathtt{tgt}) \leftarrow \mathtt{sol}(\mathtt{srce})$
  for $(d, d') \in \alpha_\beta$ *such that* $\beta_{\mathtt{srce}} \vdash \langle d$ `subStringOf` $\mathtt{sol}(\mathtt{srce})\rangle$ **do**
    $\mathtt{sol}(\mathtt{tgt}) \leftarrow \mathtt{replace}(d, d', \mathtt{sol}(\mathtt{tgt}))$
  end
  return $\mathtt{sol}(\mathtt{tgt})$
end

Fig. 5. An implementation of analogical transfer step AT2.

recalling that $\mathtt{sol}(\mathtt{srce}) =$ `"irish whisky"`. Applying the algorithm of figure 4, the computation of $\widehat{\alpha_\beta}$ and $\beta_{\mathtt{gen}}$ gives:

$$\begin{aligned}\widehat{\alpha_\beta} = \{&(\mathtt{irishWhisky}, \mathtt{tequila}), (\mathtt{ireland}, \mathtt{?x}), (\mathtt{"irish"}, \mathtt{?y}),\\ &(\mathtt{"irish\ coffee"}, \mathtt{?solTgt}), (\mathtt{coffee}, \mathtt{coffee}), (\mathtt{"coffee"}, \mathtt{"coffee"})\}\\ \beta_{\mathtt{gen}} = \{&\langle \mathtt{tequila\ origin\ ?x}\rangle, \langle \mathtt{?x\ englishAdjective\ ?y}\rangle,\\ &\langle \mathtt{?y\ subStringOf\ ?solTgt}\rangle, \langle \mathtt{coffee\ nameInEnglish\ "coffee"}\rangle,\\ &\langle \mathtt{"coffee"\ subStringOf\ ?solTgt}\rangle\}\end{aligned}$$

From this, the following SPARQL query is built and executed on `DK`:

SELECT `?x ?y` WHERE $\{$`tequila origin ?x . ?x englishAdjective ?y`$\}$

Assuming the result `R` contains the only assignment $A =$ $\{$`?x` $\leftarrow$ `mexico`, `?y` $\leftarrow$ `"mexican"`$\}$, it comes that $P_{\alpha_\beta \beta_{\mathtt{tgt}}} = \{(\alpha_\beta, \beta_{\mathtt{tgt}})\}$ with

$$\begin{aligned}\alpha_\beta = \{&(\mathtt{irishWhisky}, \mathtt{tequila}), (\mathtt{ireland}, \mathtt{mexico}), (\mathtt{"irish"}, \mathtt{"mexican"}),\\ &(\mathtt{"irish\ coffee"}, \mathtt{?solTgt}), (\mathtt{coffee}, \mathtt{coffee}), (\mathtt{"coffee"}, \mathtt{"coffee"})\}\\ \beta_{\mathtt{tgt}} = \{&\langle \mathtt{tequila\ origin\ mexico}\rangle, \langle \mathtt{mexico\ englishAdjective\ "mexican"}\rangle,\\ &\langle \mathtt{"mexican"\ subStringOf\ ?solTgt}\rangle,\\ &\langle \mathtt{coffee\ nameInEnglish\ "coffee"}\rangle,\\ &\langle \mathtt{"coffee"\ subStringOf\ ?solTgt}\rangle\}\end{aligned}$$

And, finally, the solution proposed is obtained by replacing successfully (a) `"irish"` by `"mexican"` and (b) `"coffee"` by `"coffee"` in $\mathtt{sol}(\mathtt{srce}) =$ `"irish coffee"`:

$$\mathtt{"irish\ coffee"} \overset{\text{(a)}}{\longmapsto} \mathtt{"mexican\ coffee"} \overset{\text{(b)}}{\longmapsto} \mathtt{"mexican\ coffee"} = \mathtt{sol(tgt)}$$

6.4 Strategy "Generalization-specialization of dependencies"

Now, consider the example 5, of the adaptation of $\mathtt{sol}(\mathtt{srce}) =$ `"blue lagoon"` when $\alpha_\beta =$ `curaçao` $\rightsquigarrow$ `indianTonic` with the same $\beta_{\mathtt{srce}}$ as in example 1 (cf. §5), and assuming that `DK` gives no color to Indian tonic (i.e., there is no triple of the form

$t = \langle$`indianTonic color c`$\rangle$ such that $\texttt{DK} \vdash t$), the adaptation strategy of section 6.3 fails. However, it is assumed that

$$\texttt{DK} \vdash \left\{ \begin{array}{c} \langle\texttt{indianTonic taste bitter}\rangle, \langle\texttt{indianTonic texture sparkling}\rangle, \\ \langle\texttt{bitter inEnglish "bitter"}\rangle, \langle\texttt{sparkling inEnglish "sparkling"}\rangle, \\ \langle\texttt{color subp hOP}\rangle, \langle\texttt{taste subp hOP}\rangle, \langle\texttt{texture subp hOP}\rangle \end{array} \right\}$$

meaning that Indian tonic is bitter and sparkling, and that color, taste and texture are organoleptic properties (`hOP` is an abbreviation for `hasOrganolepticProperty`). Therefore, the adaptation strategy described in section 6.3 can be applied with a slight modification: it is sufficient to replace in $\beta_{\texttt{gen}}$ the triple $\langle$`indianTonic color ?x`$\rangle$ by $\langle$`indianTonic hOP ?x`$\rangle$, which is more general according to `DK`.

One way to address this problem is to replace all the resources and literals of $\beta_{\texttt{srce}}$—including the predicates—by variables, with the exception of the ones matched by $\alpha_{\texttt{pb}}$ (i.e., `indianTonic` in the example). This would lead in the example to

$$\alpha_{\texttt{pb}}(\beta_{\texttt{srce}}) = \{\langle\texttt{indianTonic color blue}\rangle, \langle\texttt{color inEnglish "blue"}\rangle, \langle\texttt{"blue" subStringOf sol(srce)}\rangle\}$$

generalized into

$$\beta_{\texttt{gen}} = \{\langle\texttt{indianTonic ?p1 ?x}\rangle, \langle\texttt{?x ?p2 ?y}\rangle, \langle\texttt{?y ?p3 ?solTgt}\rangle\}$$

However, we choose to discard this approach because it may give too many results and since it is based on a too shallow semantics. For example, $\texttt{sol(tgt)} = \texttt{"food lagoon"}$ would be justified by the assignment $\{\texttt{?p1} \leftarrow \texttt{subc}, \texttt{?x} \leftarrow \texttt{food}, \texttt{?p2} \leftarrow \texttt{inEnglish}, \texttt{?y} \leftarrow \texttt{"food"}\}$.

Another way to address this problem is to search in the domain knowledge for triples for building $\beta_{\texttt{gen}}$ that are *similar* to $\alpha_\beta(\beta_{\texttt{srce}})$. This can be likened to the retrieval issue in CBR, which can be implemented by a least generalization of the query (see, e.g., [5]). A similar idea is proposed here. It consists in making a best-first search in a space of dependencies β such that:

- The initial state β_0 corresponds to the $\beta_{\texttt{gen}}$ as it is computed in the strategy of section 6.3.
- The successors of a state consists in making a generalization of one of its triples. The following generalization operators can be considered: replace a class (resp., a property) by a direct superclass (resp., direct superproperty) in `DK`, replace a resource or a literal by a variable, etc. A cost function must be associated to generalization operators, in order to choose the least costly generalization.
- A final state β is such that the SPARQL query associated with it gives a nonempty set of results.

Once a final state β is found, the rest of the algorithm of section 6.3 can be applied with $\beta_{\texttt{gen}} = \beta$.

Back to the example, it comes:

$$\beta_0 = \{\langle\texttt{indianTonic color ?x}\rangle, \langle\texttt{?x inEnglish ?y}\rangle, \langle\texttt{?y subStringOf ?solTgt}\rangle\}$$

In the first triple, color can be generalized into hOP (since DK $\vdash \langle$color subp hOP$\rangle$), giving

$$\beta = \{\langle \texttt{indianTonic hOP ?x}\rangle, \langle \texttt{?x inEnglish ?y}\rangle, \langle \texttt{?y subStringOf ?solTgt}\rangle\}$$

β is a final state since exec(Q, DK) $\neq \emptyset$ for

$$\texttt{Q} = \text{SELECT } \texttt{?x ?y} \text{ WHERE } \{\texttt{indianTonic hOP ?x . ?x inEnglish ?y}\}$$

Indeed, exec(Q, DK) $= \{A_1, A_2\}$ where $A_1 = \{\texttt{?x} \leftarrow \texttt{bitter}, \texttt{?y} \leftarrow \texttt{"bitter"}\}$ and $A_2 = \{\texttt{?x} \leftarrow \texttt{sparkling}, \texttt{?y} \leftarrow \texttt{"sparkling"}\}$, leading to the two expected solutions: "bitter lagoon" and "sparkling lagoon".

Therefore this strategy consists in finding the minimal generalization β of the initial dependency β_0 and then in specializing β into $\beta_{\texttt{tgt}}$'s thanks to SPARQL querying on DK, hence the name of the strategy.

6.5 Composing strategies when the matching is complex

When the matching $\alpha_{\texttt{pb}}$ is complex, it can be written $\alpha_{\texttt{pb}} = \alpha^q_{\texttt{pb}} \circ \alpha^{q-1}_{\texttt{pb}} \circ \ldots \circ \alpha^1_{\texttt{pb}}$, with $q \geqslant 2$. The idea is then to apply in sequence the strategies associated with simple matchings. For example, for sol(srce) = "irish coffee", $\alpha^1_{\texttt{pb}} = \texttt{irishWhisky} \rightsquigarrow \texttt{tequila}$, $\alpha^2_{\texttt{pb}} = \texttt{coffee} \rightsquigarrow \texttt{hotChocolate}$, the strategy of section 6.3 can be applied twice to give the name sol(tgt) = "mexican hot chocolate". This adaptation is an application of the adaptation based on reformulations and similarity paths (see e.g. [8]).

Another example consists in substituting in the "Blue Lagoon" recipe all the ingredients by sparkling water, in the order curaçao, vodka and lemon juice, giving birth to the name "the new virgin sparkling lagoon" for a glass of sparkling water, which can arguably be considered as the result of a creative naming process!

7 Discussion

Among the 20 examples listed in the first phase of this study (cf. the sample presented in section 4), 13 are modeled in the strategies above: 1 in §6.1, 1 in §6.2, 9 in §6.3 and 2 in §6.4[7] (0 in §6.5). 5 of the remaining ones corresponds to a strategy consisting in adding or substituting a qualifier to the source recipe name when a new ingredient is added or replaces an ingredient that has no connection with the source recipe name. For example, "gin fizz" becomes "silver fizz" when an egg white is added to the recipe. Finally, 2 examples are not covered because they would require a more complex case representation, for example, "tequila sunrise" becomes "tequila sunset" partially because of the change in the order of the preparation steps. These figures do not

[7] Actually, the 9 of §6.3 could also be counted as modeled by §6.4: the second strategy generalizes the latter.

constitute statistically significant information but give some ideas on how the examples has led to strategies and how they can be used to guide future work.

Apart from the ad hoc strategies, the analogical transfer strategies presented in the previous section corresponds to a scheme of modifying (by generalization) a dependency so that it becomes consistent with the target context (under CWA). Modifying a case until it reaches consistency with the target problem wrt the domain knowledge is what revision-based adaptation (RBA) does [3]. Therefore, though RBA has not been a useful guideline for starting this research, it could be used to re-describe this contribution and to go one step further, in order to examine formal properties of the analogical transfer and to propose new strategies. Actually, in previous studies, RBA was used in order to modify the *solution* $\texttt{sol}(\texttt{srce})$ of the retrieved case (which is uneasy to formalize when solutions are strings), whereas RBA could be used as a tool to modify the *dependency* $\beta_{\texttt{srce}}$ within the AT process.

Following this idea, the analogical transfer amounts to travel in a dependency space structured by modifications (only generalizations in the examples given in this paper), with a good choice of the travel costs. The strategy described in sections 5 and 6.3 works with a constant modification that turns the edges of the RDFS graph into variables but does not modify the properties that label the edges of this graph. This could be understood as the fact that the cost of the former generalizations is much lower than the cost of the latter ones. To justify this, it is considered that the properties (e.g., `color`) are more abstract descriptors than other resources (e.g., `blue`, `yellow`). According to the heuristic saying that it is better to modify a concrete descriptor than an abstract one, this is justified. This heuristic principle has been defended for a long time in the analogical reasoning community [6].

8 Conclusion and future work

Starting from the applicative problem of cocktail name adaptation, this paper describes a research work on strategies for analogical transfer in the context of the representation language RDFS. If some proposed strategies are application-dependent, it is claimed that other ones can be applied—or adapted—to a larger framework. Indeed, they match the principles described in some related work about analogical transfer (e.g., [6] and [14]) while proposing an approach having profit of the standard RDFS as well as on associated tools (RDFS SPARQL engines, RDF stores).

The operationality of this work is demonstrated by a first prototype in Python that covers some of the strategies. However, some work remains to be done to cover all of them, in particular the one based on generalization-specialization of dependencies, which constitutes an ongoing work. Furthermore, new strategies have to be developed (the strategies presented here covers the majority of the examples but not all of them: 13 on 20) and a way to control the application of strategies should be designed.

A first direction of future work aims at addressing two current limitations of the approach. First, there is an important workload for acquiring dependencies $\beta_{\texttt{srce}}$, which is currently done manually. Second, in order to get more relevant results, it is important to have more triples in the domain knowledge. In order to address these issues, it is planned to interrogate the Linked Open Data (LOD), i.e., a huge cloud of RDF and

RDFS bases freely accessible on the Web (DBPedia, a base of the LOD, contains about 3 billions triples). For example, there are tools that enable to find paths in the LOD from a resource to another one (see, e.g., [13]), and such a tool could be used to find a link from an ingredient name of a cocktail recipe to a word occurring in the name of the cocktail or—more generally—from a problem $\texttt{srce}$ to a solution $\texttt{sol}(\texttt{srce})$ of this problem. The union of such paths would constitute the dependency $\beta_{\texttt{srce}}$ and the domain knowledge should contain at least the union of all the $\beta_{\texttt{srce}}$'s.

References

1. Carbonell, J.G.: Derivational analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In: Machine Learning. Volume 2. Morgan Kaufmann, Inc. (1986) 371–392
2. Ceci, F., Weber, R.O., Gonçalves, A.L., Pacheco, R.C.S.: Adapting Sentiments with Context. In: Case-Based Reasoning Research and Development. Springer (2015) 44–59
3. Cojan, J., Lieber, J.: Applying Belief Revision to Case-Based Reasoning. In Prade, H., Richard, G., eds.: Computational Approaches to Analogical Reasoning: Current Trends. Volume 548 of Studies in Computational Intelligence. Springer (2014) 133 – 161
4. Cordier, A., Dufour-Lussier, V., Lieber, J., Nauer, E., Badra, F., Cojan, J., Gaillard, E., Infante-Blanco, L., Molli, P., Napoli, A., Skaf-Molli, H.: Taaable: a Case-Based System for personalized Cooking. In Montani, S., Jain, L.C., eds.: Successful Case-based Reasoning Applications-2. Volume 494 of Studies in Computational Intelligence. Springer (2014) 121–162
5. Gaillard, E., Infante-Blanco, L., Lieber, J., Nauer, E.: Tuuurbine: A Generic CBR Engine over RDFS. In: Case-Based Reasoning Research and Development. Volume 8765., Cork, Ireland (September 2014) 140 – 154
6. Gentner, D.: Structure-Mapping: A Theoretical Framework for Analogy. Cognitive science **7**(2) (1983) 155–170
7. Jalali, V., Leake, D.: CBR Meets Big Data: A Case Study of Large-Scale Adaptation Rule Generation. In: Case-Based Reasoning Research and Development. Springer (2015) 181–196
8. Melis, E., Lieber, J., Napoli, A.: Reformulation in Case-Based Reasoning. In Smyth, B., Cunningham, P., eds.: Fourth European Workshop on Case-Based Reasoning, EWCBR-98. LNCS 1488, Springer (1998) 172–183
9. Müller, G., Bergmann, R.: Workflow streams: A means for compositional adaptation in process-oriented CBR. In Lamontagne, L., Plaza, E., eds.: Case-Based Reasoning Research and Development, ICCBR-2014. Volume 8765 of Lecture Notes in Artifichal Intelligence. Springer, Cork, Ireland (2014) 315–329
10. Müller, G., Bergmann, R.: CookingCAKE: A Framework for the adaptation of cooking recipes represented as workflows. (2015)
11. Riesbeck, C.K., Schank, R.C.: Inside Case-Based Reasoning. Lawrence Erlbaum Associates, Inc., Hillsdale, New Jersey (1989)
12. Sowa, J.F.: Conceptual Structures: Information Processing in Mind and Machine. Addison Wesley, Reading, Massachusetts (1984)
13. Tiddi, I., d'Aquin, M., Motta, E.: Dedalo: Looking for clusters explanations in a labyrinth of linked data. In: ESWC-2014. LNCS, Springer (2014)
14. Winston, P.H.: Learning and reasoning by analogy. Communications of the ACM **23**(12) (1980) 689–703