Text adaptation using formal concept analysis

Valmi Dufour-Lussier, Jean Lieber, Emmanuel Nauer, Yannick Toussaint

Text Adaptation Using Formal Concept Analysis*

Valmi Dufour-Lussier, Jean Lieber, Emmanuel Nauer, and Yannick Toussaint

LORIA—UMR 7503 (CNRS, INRIA, Nancy-Université)
BP 239, 54506 Vandœuvre-lès-Nancy, France
{valmi.dufour,jean.lieber,emmanuel.nauer,yannick.toussaint}@loria.fr

Abstract. This paper addresses the issue of adapting cases represented by plain text with the help of formal concept analysis and natural language processing technologies. The actual cases represent recipes in which we classify ingredients according to culinary techniques applied to them. The complex nature of linguistic anaphoras in recipe texts make usual text mining techniques inefficient so a stronger approach, using syntactic and dynamic semantic analysis to build a formal representation of a recipe, had to be used. This representation is useful for various applications but, in this paper, we show how one can extract ingredient–action relations from it in order to use formal concept analysis and select an appropriate replacement sequence of culinary actions to use in adapting the recipe text.

Key words: formal concept analysis, natural language processing, text mining, textual case-based reasoning

1 Introduction

A case retrieved by a case-based reasoning (CBR) system in order to solve a given problem may need adaptation in order to fit in. Adapting a textual case may be as simple as replacing all the occurrences of a word with another word, but one could want to do better. Contestants in the Computer Cooking Contest[1] (CCC) use case-based reasoning with a recipe book as a case base to propose ingredient substitutions as a solution to cooking problems (adapting a recipe to given constraints) but so far are not making modifications to the recipe text.

This paper shows that using a method based on text mining and machine learning, namely formal concept analysis (FCA), can be of great use for text adaptation. Ingredient preparation *prototypes* are found and used to adapt a recipe. Adapting a recipe by replacing ingredient α with β implies finding out actions performed on α and replacing them with actions performed on β.

* The authors would like to thank the reviewers for their helpful comments, as well as Zeeshan Ahmed for his previous work on a related topic and Jenia Berzak, Le Dieu Thu, and Ripan Hermawan for letting us use their corpus of part-of-speech-tagged recipe texts.

[1] http://vm.liris.cnrs.fr/ccc2010

The work was achieved within the Taaable project [4, 8] and focuses on text adaptation. Taaable is a textual case-based cooking system that participated in the first and second (as WikiTaaable) CCC. It is built around a case-based inference engine using a (minimal) propositional representation of recipes, a set of known acceptable substitutions, an ontology of ingredients used to build new substitutions on the fly, and a cost function to select the best adaptation for a problem.

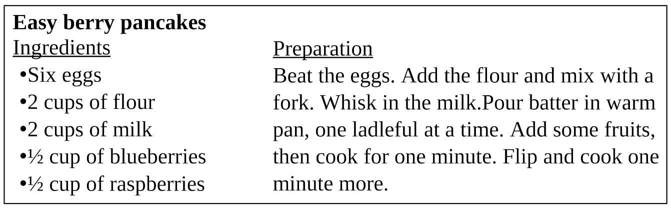
In this paper, we shall argue for a more thorough formal representation of recipes, and show how it can be built with natural language processing (NLP) techniques and used with FCA towards a more significant adaptation function.

Presupposing that Taaable is able to suggest a recipe from its case base along with some substitution operations that consist in replacing a given ingredient by another given ingredient, our system is able to find an adequate sequence of actions for the new ingredient and modify the recipe text accordingly.

While selecting texts with FCA and reusing them in the adaptation stage of a textual CBR system is to our knowledge a novel approach, it fits within a trend towards the maximal reuse of existing text in providing textual solutions to problems (arguably initiated by [15, 14]). Using FCA for information retrieval or CBR in itself is not a totally new idea either (see for instance [7, 17] for retrieval, and [10, 9] for CBR).

In Sect. 2, we describe the kind of formal representation we expect to create from recipe texts and the process to translate texts into this representation. Then in Sect. 3 we show how FCA is used to adapt recipes, and we detail the algorithms we developed as well as the strategy we used to generate new texts. Finally we discuss our results and future work in Sects. 4 and 5.

2 Linguistic Processing of Recipes



Easy berry pancakes

Ingredients	Preparation
•Six eggs •2 cups of flour •2 cups of milk •½ cup of blueberries •½ cup of raspberries	Beat the eggs. Add the flour and mix with a fork. Whisk in the milk.Pour batter in warm pan, one ladleful at a time. Add some fruits, then cook for one minute. Flip and cook one minute more.

Fig. 1: A sample recipe text.

The main idea guiding this work is that some "common uses" of each ingredient, that we call *prototypes,* can be extracted from the case base and used in adapting texts. If for instance we want to substitute *zucchini* with *aubergine* in a recipe, it would be more convenient to prepare the aubergine as done in some aubergine recipes instead of blindly applying the same steps as for preparing the zucchini. A prototype is understood as a sequence of actions applied to an ingredient. To extract it, we need a formal representation of the recipe text. The same linguistic processing that changes a text in its representation is performed on the source recipe (the recipe to be adapted) and on all the recipes of the case base. Those formal representations are then passed on to data mining algorithms to extract the prototypes. The complete process will be illustrated on the recipe in Fig. 1, yielding the representation shown in Fig. 2.

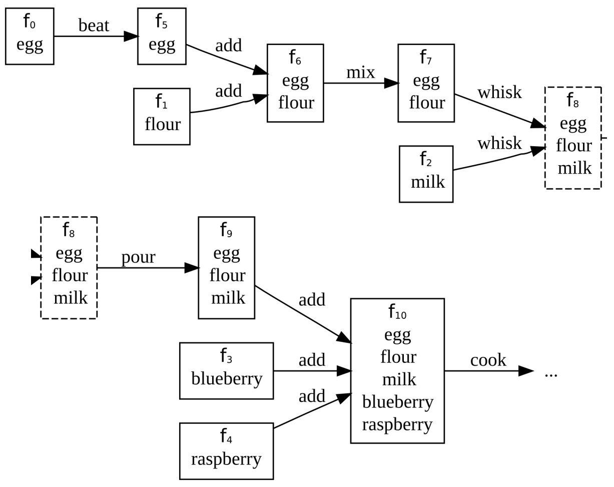


Fig. 2: Tree representation of the recipe of Fig. 1.

2.1 Representing Recipes as Trees

Recipes are procedural texts composed of a sequence of actions through which different ingredients are progressively combined in order to obtain one final product, the dish. Each culinary action takes what we call *food components* in entry (as its *arguments*) and produces some other food component in return.

To adapt a recipe, it can help to divide it in smaller "parts" so that some parts can be replaced by new ones. Therefore the formal representation of a recipe must make it easy to identify the different stages in a recipe and the "regularities" across a set of recipes. Viewing actions as functions, it seems only natural to model recipes as trees. Alternatively this can be seen as taking a rather extreme stance in regards to Asher's theory of dynamic semantics [1], considering each verb as simultaneously a destruction and a construction verb (applied onto a food component). Some situations would make trees inappropriate, such as when whole eggs are split between white and yolk. We didn't take this into account in this work, but we think our approach could easily generalised to directed acyclic graph representations.

In a recipe tree, leaves are food components corresponding to the raw ingredients, the root is the finished dish, and the other nodes are the subsequent states of various food components. Trees are labelled (each node has a unique label ℓ) and a function $\mathcal{I}(\ell)$ is defined giving the set of ingredients that went into the food component represented by ℓ. For instance in Fig. 2, giving an example of a very simple recipe tree, "f_6" is a node corresponding to a food component such that $\mathcal{I}(\mathrm{f}_6) = \{\mathrm{egg}, \mathrm{flour}\}$.

This tree structure is what the data mining process is applied to. However it is also needed to solve some of the linguistic problems of recipe texts. The tree is built iteratively: actions found in the text are treated one after the other, each connecting a node to the tree, a process that requires using the information already present in the partially built tree.

2.2 Why Recipe Texts Are Different

While recipe texts have the advantage of exhibiting little fanciness, there exist specific difficulties inherent to their procedural nature:

1. They heavily make use of sentence structures such as imperative clauses that are rare in most other texts, making tools based on machine learning using generic corpora inefficient;
2. They massively exhibit a little-studied linguistic phenomenon known as *evolutive anaphora* wherein a word in a text may be used to refer to an object that exists at some given time and does not (yet or anymore) at some other, requiring a special strategy to find out what this word can refer to at any given moment *e.g.* "mix flour, eggs, and milk; pour *the batter*";
3. In order to avoid tedious repetitions, they usually omit syntactic arguments of verbs when they seem obvious, requiring a strategy to first determine whether a word is missing, and finding out what this word should have been, *e.g.* "cook potatoes; when done, add milk [implicitly: *to potatoes*]" (this is a type of *grammatical anaphora*).

2.3 The Toolchain

The first few steps of the linguistic analysis can be solved using common, well-researched natural language techniques. While we cannot use available annotated corpora as much as we normally would, we still managed to obtain a small corpus of about a hundred recipes annotated with the part-of-speech (*e.g.* verb, noun, adjective) of each word, which was sufficient to train an error-driven, context-sensitive, transformation-based tagger [6] with an accuracy $a > .90$, well below the state of the art for regular texts, but sufficient for a prototype implementation.[2] The other preliminary steps (tokenization, clause segmentation, chunking) were implemented using hand-crafted regular expressions.[3]

At this stage we know which words are verbs (and thus actions) and we are able to identify their syntactic arguments, which should be sufficient to get on with the task of building a formal tree representation of a recipe.

2.4 From Text to Tree

The problems described in paragraphs 2.2(2) and 2.2(3) cannot be solved at the sentence level. On the other hand, if they are neglected, the final representation of a recipe will not be a tree, but a set of trees, each representing a part of the recipe, that cannot be connected. A specific post-processing step was developed to handle those.

The idea is that at the beginning of the recipe, all ingredients are available and the very first action will take some of them to produce a new food component. In the same way, at any stage of a recipe, there are certain food components available and the next action picks some of them and produces a new one.

The set of food components available for culinary actions is called *domain.* The initial domain $\mathcal{D}_0$ contain one food component for each ingredient from the recipe listings. The domain changes after each action is performed, hence the domain after t actions, noted $\mathcal{D}_t$, is used to identify the arguments of the $t+1^{\text{th}}$ action.

In the recipe of Fig. 1, the initial domain will look something be:

$$\begin{aligned}\mathcal{D}_0 =&\{\mathrm{f}_0, \mathrm{f}_1, \mathrm{f}_2, \mathrm{f}_3, \mathrm{f}_4\},\\ \mathcal{I}(\mathrm{f}_0) =& \{\mathrm{egg}\},\\ \mathcal{I}(\mathrm{f}_1) =& \{\mathrm{flour}\}, \mathrm{etc.}\end{aligned}$$

When an action that takes a single argument is encountered, it creates an arrow with the action name and creates a new node. It also modifies the domain, such that for a verb like "beat X":

$$\begin{aligned}\mathcal{D}_t =&(\mathcal{D}_{t-1}\backslash\{X\}) \cup \{\ell\},\\ \mathcal{I}(\ell) =& \mathcal{I}(X)\ ,\end{aligned} \tag{1}$$

where ℓ is a new label. For instance the first sentence in the example recipe is "beat the eggs", which would have the effect of creating a new "egg" food component (see Fig. 3):

$$\begin{aligned}\mathcal{D}_1 =&(\mathcal{D}_0\backslash\{\mathrm{f}_0\}) \cup \{\mathrm{f}_5\},\\ \mathcal{I}(\mathrm{f}_5) =& \mathcal{I}(\mathrm{f}_0)\ .\end{aligned}$$

[2] Parts-of-speech tend to be even more ambiguous than usual in recipe texts because of words such as "cream" or "salt" that can be both nouns or verbs, so an approach based on a dictionary, even if it were a domain-specific dictionary, would be even less effective.

[3] For instance, the regular expression pattern for matching a noun phrase is "$\mathrm{N}'((\mathrm{Comma}\ \mathrm{N}')^\star(\mathrm{Comma}|\mathrm{Conjunction})^{\{1,2\}}\mathrm{N}')^?$", with "N′" matching "$\mathrm{Predeterminer}^?\ \mathrm{Determiner}^?\ \mathrm{Adjective}^\star\ \mathrm{Noun}^+$".

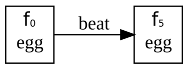


Fig. 3: Beaten eggs.

As for actions taking several arguments, they may have either a "union" semantics, like "add X to Y", creating a node with multiple edges,

$$\begin{aligned}\mathcal{D}_t =& (\mathcal{D}_{t-1} \backslash (\{X\} \cup \{Y\})) \cup \{\ell\}, \\ \mathcal{I}(\ell) =& \mathcal{I}(X) \cup \mathcal{I}(Y) \ ,\end{aligned} \tag{2}$$

or a "complement" semantics, like "remove X from Y":

$$\begin{aligned}\mathcal{D}_t =& (\mathcal{D}_{t-1} \backslash \{Y\}) \cup \{\ell\}, \\ \mathcal{I}(\ell) =& \mathcal{I}(Y) \backslash \{X\} \ ,\end{aligned} \tag{3}$$

where ℓ is a new label. For instance, the next sentence in the example recipe is "Add the flour", which is a union action (see Fig. 4):

$$\begin{aligned}\mathcal{D}_2 =& (\mathcal{D}_1 \backslash (\{\mathrm{f}_1\} \cup \{\mathrm{f}_5\})) \cup \{\mathrm{f}_6\}, \\ \mathcal{I}(\mathrm{f}_6) =& \mathcal{I}(\mathrm{f}_1) \cup \mathcal{I}(\mathrm{f}_5) \ .\end{aligned}$$

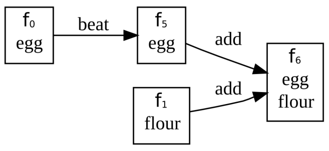


Fig. 4: Some batter.

A *subcategorization dictionary* of actions tells which types of arguments are required by each action in order to know whenever one is missing. In that case, the last node added to the tree is assumed to be the missing argument, thus an edge is created from this node to the new node. This is how one can infer that it is to eggs that flour gets added (cf. Fig. 4).

The set-theoretical notation we used for food components' ingredients is useful to resolve the anaphoras. The two most frequent cases are presented below.

Existential References. Expressions such as "beef mixture" refer to some food component that contain at least one specific ingredient, in this case beef, that may have been mixed with others. It is therefore necessary to search the domain for a food component containing this ingredient. A target set $\mathcal{T}$ of ingredients expected in the food component is thus defined and used with a simple operation to retrieve the food component being referred to by the expression:

$$x \in \mathcal{D} : \exists i . i \in \mathcal{I}(x) \wedge i \in \mathcal{T} \ . \tag{4}$$

In the case of "beef mixture", this is trivial: $\mathcal{T} = \{\text{beef}\}$. But some cases are more subtle, such as words similar to "batter". We measured in a corpus of recipes that the food component referred to by the word "batter" contains either the ingredients egg or flour in over 99% of cases. To find the food component referred to by this word, we will thus use $\mathcal{T} = \{\text{egg}, \text{flour}\}$. Thanks to this the "Pour batter" instruction of the recipe can be dealt with. Before dealing with this instruction, the representation looks like Fig. 5, making it is obvious which of the three food components in the domain is the one the word "batter" refers to.

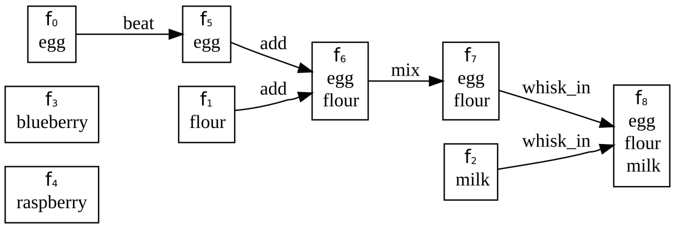


Fig. 5: Where is the batter?

Universal References. Other expressions obviously refer to a set of ingredients that belong to a common class. Such is the case for the word "fruits" when the ingredients listings of a recipe do not actually contain any "fruit" elements, but do contain blueberries and raspberries. Since an ontology of ingredients already exists in TAAABLE, it is used to retrieve the set of food component being referred to by their class name. In any given recipe, the set of food components referred to by the word "fruit" can be defined as:

$$\{x \in \mathcal{D} : \forall i . i \in \mathcal{I}(x) \rightarrow i \sqsubseteq \text{Fruit}\} \ ,$$

where "$i \sqsubseteq$ Fruit" means that ingredient i is a subclass of the "Fruit" class. So when we process the instruction "Add some fruits" in our recipe, we know that it refers to $\{\mathrm{f}_3, \mathrm{f}_4\}$.

3 Adapting Recipes with Formal Concept Analysis

At this stage, with recipes formalized as trees, for each ingredient in each recipe, there exists a path (a sequence of actions) between this ingredient and the final state of the recipe—the dish.

The adaptation process can now be redefined according to this structure. We consider that adapting the preparation with respect to an ingredient substitution consists in replacing a subtree corresponding to the preparation of an ingredient in the retrieved recipe with a subtree that is suitable for the substitution ingredient.

The questions that need to be dealt with now are the selection of a subtree to replace and a subtree to replace it with. This will be demonstrated with an example using genuine recipes from a past CCC recipe base. Suppose the user wants an aubergine coleslaw, and the system retrieved a zucchini coleslaw recipe and suggested to replace zucchini with aubergine. We need to find the instance of aubergine use in the available recipes that is the closest to the way the zucchini is used in the zucchini coleslaw. FCA provides a conceptualization of the different ways of preparing an ingredient, so that the concept closest to the zucchini can be easily identified.

3.1 Extracting the Relevant Subtree

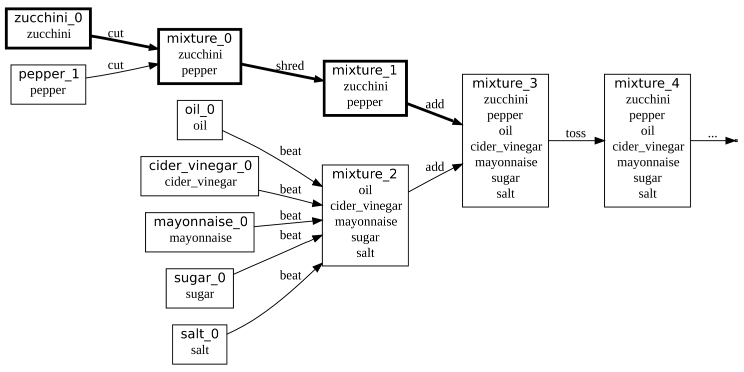


Fig. 6: Relevant "zucchini" subtree identified in bold.

We now need to extract the subtree referring to any ingredient of particular interest from the representation (this will be done both for the substituted and the substitution ingredient). Three types of actions are considered: actions applied to an ingredient alone, actions applied to many ingredients in parallel, and actions applied to many ingredients together. The distinction between the second and third types is important: while peeled apples and pears for instance are still distinctively apples *and* pears, apple and pears cooked together make up a mixture that has little to do with the original ingredients, and to which a wholly different range of actions may be applied. Additionally, if all the information is taken, the space gets too wide and there is not enough data concentration to allow for efficient learning.

Because no linguistic clues are available to help classify actions between the second and the third category, we simply use a list of possible actions with their most likely category. Considering now a genuine recipe from the case base, represented in Fig. 6, "toss" is an action of the third category hence the actions applied to zucchini are considered as relevant up to "add", making the zucchini prototype: "zucchini $\xrightarrow{\text{cut}} \cdots \xrightarrow{\text{shred}} \cdots \xrightarrow{\text{add}}$". When making the substitution later on, the zucchini prototype will therefore be detached from "mixture_3" and the aubergine prototype chosen in Sect. 3 will be reattached at the same point.

3.2 Formal Concept Analysis

FCA [11] takes as entry a binary table (such as the one shown in Table 1) describing a binary relation between objects ("aubergine_509", "aubergine_667"...) and attributes ("add", "arrange"...), called a *formal context,*. Formally, considering a formal context $\mathbb{K} = \langle \mathcal{O}, \mathcal{A}, I \rangle$, where $\mathcal{O}$ is a set of objects, $\mathcal{A}$ is a set of attributes, and $I \subseteq \mathcal{O} \times \mathcal{A}$ is a binary relation such that $\langle o, a \rangle \in I$ means that object o owns attribute a, its Galois connection is defined by the following functions: $f : 2^{\mathcal{O}} \to 2^{\mathcal{A}}$ and $g : 2^{\mathcal{A}} \to 2^{\mathcal{O}}$, where a subset of objects is mapped to the subset of attributes it owns in common, and reciprocally.Formal concepts are the $O \times A$ pairs that are closed under the Galois connection, creating the concept lattice $\mathfrak{L}(\mathbb{K})$.

Each node of a lattice such as the one shown in Fig. 7 represents a formal concept, which is a pair $\langle O, A \rangle$ where O (the *extent*) is the set of objects owning all the attributes in A (the *intent*) and A is the set of attributes owned by all the objects in O. The lattice is partially ordered following the inclusion of the extent. Any concept whose extent is included in the extent of another concept is drawn below and connected to the latter—which is said to *subsume* it.

	add	arrange	bake	beat	blend	boil	break	...
aubergine_509								
aubergine_667	×		×		×			
aubergine_981		×						
aubergine_1030			×					
aubergine_1387			×					
...								
zucchini	×							

Table 1: The formal context for the query space.

Since an ingredient's mode of preparation is characterized by the culinary actions applied onto it, the formal context that is generated uses culinary actions applied to aubergines as attributes, with each aubergine recipe corresponding to one object. The formal context, which in this case (with a tiny recipe base to maintain lattice readability) has 15 objects and 55 attributes, will thus look as in Table 1 with an object corresponding to the zucchini recipe merged along with the relevant culinary actions. The resulting lattice is shown in Fig. 7.

Formal concept number 1, called “Top”, is the maximum of the lattice. Its extent is the set of all objects and its intent is the set of the attributes shared by all objects (in this case, the empty set). Concept number 58, called “Bottom”, is the minimum of the lattice. Its intent is the set of all attributes and its extent is the set of the objects that share all attributes (there again the empty set). From top to bottom, the concepts are progressively more “specific”, meaning their extent contains less objects but those share more attributes. Concept number 29 is the most specific one containing `zucchini` in its extent, so it shows all attributes of this object. Concept number 16 is said to immediately subsume concept number 29: it is the most specific concept having `zucchini` as well as other objects in its extent (in this example there is only one concept immediately subsuming concept number 29, but there could be more). From a certain point of view, those objects are the ones that are the most closely related to `zucchini`, insofar as they are the ones having the most attributes in common with it.

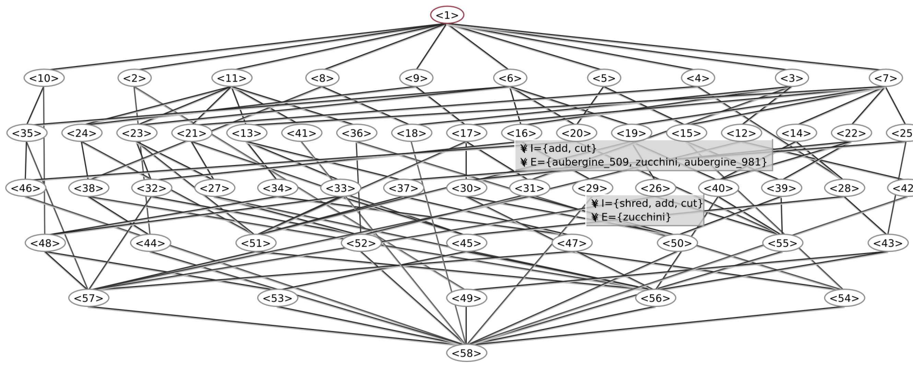


Fig. 7: The 58-concept query lattice.

3.3 Answering Queries in a Lattice

In the fashion of Carpineto [7], lattice-based ranking is used to retrieve an adaptation prototype. If one has a recipe with zucchini (say a zucchini coleslaw) and wants to replace this zucchini with aubergine (because they want an aubergine coleslaw), a first step will be to find an aubergine prototype that is compatible with the way the zucchini is prepared, possibly including steps specific to aubergines and excluding steps specific to zucchini. It makes sense to view the zucchini recipe as a query in the document space of aubergine recipes.

Given the formal concept of which our query is part of the *proper* or *reduced extent* (the “lowest”, or most specific concept in which the query appears), the candidate answers are selected from the (full) extent of this concept itself, or of the concepts immediately subsuming it, minus the query object. This heuristic is different from Carpineto's (who searches the whole set of subsuming concepts) because the goal here is to reduce the adaptation effort required, hence minimize the difference between the query's and the selected object's attributes. In this very case, the recipe that has the least differences between the attributes of the zucchini z and its replacement aubergine a is:

$$\arg\min_a \left| \Big(\mathrm{Int}(\mathcal{C}_\mathrm{z}) \backslash \mathrm{Int}(\mathcal{C}_\mathrm{a})\Big) \cup \Big(\mathrm{Int}(\mathcal{C}_\mathrm{a}) \backslash \mathrm{Int}(\mathcal{C}_\mathrm{z})\Big) \right| , \tag{5}$$

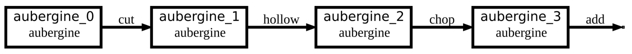
where $\mathcal{C}_\mathrm{a}$ and $\mathcal{C}_\mathrm{z}$ are the most specific concepts of a and z, and $\mathrm{Int}(\mathcal{C})$ is the intent of $\mathcal{C}$.

3.4 The Adaptation Algorithm

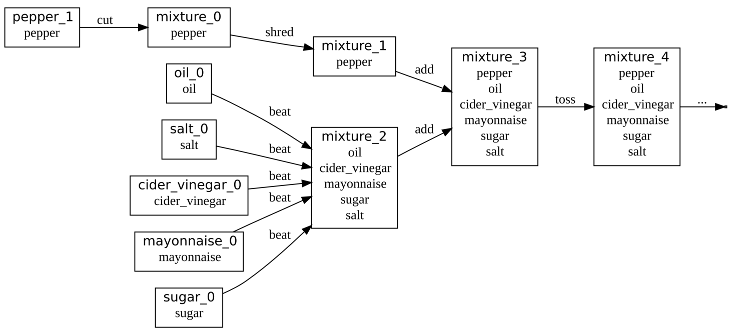
All that is left to do to complete the tree adaptation is to obtain the zucchini and aubergine subtrees using the technique described in Sect. 2.4, remove the former (keeping parts that are required for the processing of other ingredients), and merge the latter.

Existing partial-order case-based planning techniques such as [19, 13, 3] could be reused to integrate the new preparation steps along the rest of the recipe. On the other hand, not keeping the structure of the selected recipe intact would prevent us from reusing its text as is, forcing us to use natural language generation techniques and yielding results of poor textual quality, as argued in [12]. Moreover planning requires advanced knowledge of the domain: action pre- and post-conditions must be known. For instance, a given ingredient may, when heated, create froth that needs removing. In the absence of such knowledge, grafting seems a safer approach. A simpler strategy that consists into connecting the new subtree at only one point in the tree gives results that are satisfactory (as shown in Fig. 8) and allows for easier textual adaptation, at a reduced computational cost.

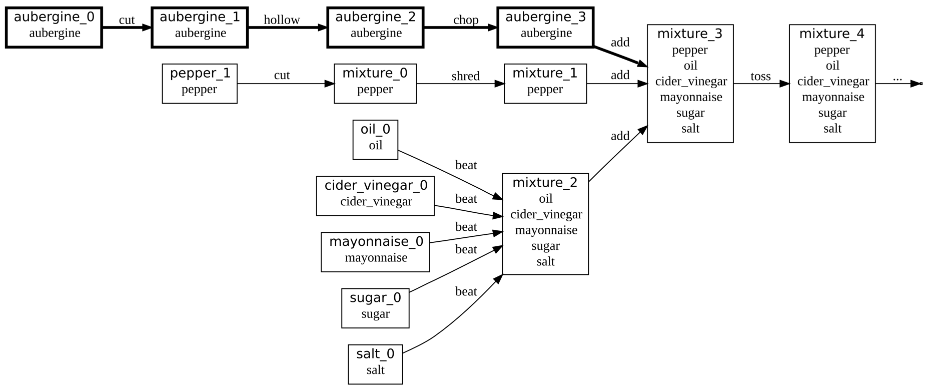
We propose an algorithm for text adaptation that is very simple and parallel to the tree adaptation process. It copies whole sentences of the selected recipe text as much as possible, causing the resulting text to have a more "natural" quality than if it has been machine generated. Actions related to zucchini are removed: if a clause refers to zucchini only it is removed altogether, otherwise only the word "zucchini" is removed (along possibly with the word "and" or a comma). Then the text related to aubergine preparation in the selected recipe is treated in the same way to remove references to all ingredients other than aubergines. This "aubergine text" is inserted at the beginning of the original text, except for the clause containing the last verb of the prototype (the one that causes it to be merged with other ingredients), which is inserted at the point where those other ingredients have been fully processed. This process is exemplified in Fig. 9. All that is then left to do is minor grammatical (*e.g.* verb agreement) and typographic (*e.g.* punctuation) adjustments.



(a) The `aubergine_981` subtree to be attached.



(b) The tree from Fig. 6 with zucchini subtree pruned.



(c) The final tree after adaptation performed.

Fig. 8: The subtree attachment process.

Cut the aubergines in half lengthwise. With a spoon, hollow out the center of each half to make a boat like shell about 1/4 inch thick. Finely chop the aubergine pulp and set it aside. Cut ~~zucchini and~~ red pepper into matchstick thin strips or shred in food processor; add the chopped aubergine pulp. set aside. In large bowl, with wire whisk or fork, beat salad oil, vinegar, mayonnaise, sugar, salt and pepper until mixed. Add vegetables; gently toss to mix well. [...]

Fig. 9: The original recipe text with removed parts stroke through and added parts (copied from the retrieved recipe) underlined.

4 Discussion and Related Work

The adaptation process gives very satisfactory results when the data mined in text was of good quality. The linguistic processing of recipes is thus the weak link. The formal representation of a recipe is usually of high quality if the text contained no spelling mistakes and no "less usual" phrase structures such as negations ("do not peel the potatoes before boiling them") or elaboration ("boil the potatoes—but peel them first"). Those phenomena could be dealt with, with more or less success, but would significantly complicate the implementation.

The worst problem though is the bad performance of the part-of-speech tagger (the module responsible for saying which words are verbs, which are nouns, etc.), given that it has access to only a very small and low quality training corpus. But such a corpus is expensive to obtain: annotators can only process about 3000 words per hours [16] whereas hundreds of thousands of words are required to achieve good performance as can be seen in [5]. Because of the reference problems described in Sect. 2.2, if one action is missed by the analyzer (because the action verb was not tagged as a verb), it is very likely that many subsequent actions applied to the same ingredient will be missed.

Circa 10% of the formal representations were actual trees, indicating that those were near-perfect and totally usable representations. As for the others, "correctness" is difficult to quantify, but we would say that the analysis is usually very good for the first few actions (the one we actually use most) and decreases as it goes on.

Evaluating a case-based reasoning system is generally very difficult, but can be done through comparisions with other similar systems, which is the very reason why the CCC workshop was created. Therefore the adaptation algorithms described in this paper are being implemented in the Taaable system to be evaluated during this year's edition of the CCC.

FCA has already been used in CBR, for instance in [9] and in [2]. In [9], the formal context is a representation of the case base (the objects are the cases and their properties are binary properties of cases). The concept lattice obtained by FCA structures the case base, and the retrieval process is done thanks to a hierarchical classification in this lattice. Moreover, the lattice is used to assist the user query formulation process. By contrast, in our work, FCA is used for adaptation: the formal context is all about the vocabulary for describing cases (the ingredients and the actions performed on ingredients). The objective of [2] is to obtain structured cases from texts, using FCA. The formal context objects are the texts and its properties are relevant terms of these texts. Contrary to our problem though, in the reports they use, one document can be understood as one case. In our case, the boundary is blurred by the presence of multiple ingredients in any recipe, meaning that not all keywords found in a given text are relevant for a given adaptation.

Many of the more interesting types of textual and non-textual cases have a structure similar to recipes: assembly instructions, user's manuals, pharmaceutical directions for preparation are all examples of "procedural" cases that have certain "preconditions" that could be lacking and thus in need of adaptation. We believe that our approach would help solve this problem.

5 Conclusion and Future Work

This paper proposes a solution to adaptation in textual case-based reasoning systems. It uses natural language processing techniques to build a rich formal representation of texts, then data mining algorithms and formal concept analysis to retrieve a text from which some parts are reused.

The formal representation we created is helpful in interesting ways at other stages of the CBR process besides adaptation. For instance, a function using some physical characteristics, such as the size or the texture, of a food component in its rightmost state of the prototype sequence could be used to assess the quality of the adaptation choices, or contribute to the adaptation cost function. Moreover, the trees could be passed to a learning algorithm to ease the recognition of certain recipe attributes: *e.g.,* the sequence of action–ingredient pairs that makes a dish a soup could be learned, a knowledge which might prove useful to the CBR process.

According to the *adaptation-guided retrieval* principle [18], a source case requiring less adaptation effort should be preferred over the others. In the current implementation of Taaable, this adaptation effort is measured using a penalization cost in the ingredient hierarchy. In the future, we will incorporate a cost related to the adaptation of the preparation: if α and β are two ingredients, the more prototypes that FCA shows they have in common, the least the adaptation effort of substituting α with β will be.

References

1. Asher, N.: Events, facts, propositions, and evolutive anaphora. In: Higginbotham, J., Pianesi, F., Varzi, A. (eds.) Speaking of events, pp. 123–150. OUP, Oxford (2000)
2. Asiimwe, S., Craw, S., Wiratunga, N., Taylor, B.: Automatically acquiring structured case representations: The SMART way. In: Applications and Innovations in Intelligent Systems XV. pp. 45–58. Springer (2007)
3. noz Avila, H.M., Weberskirch, F.: Planning for manufacturing workpieces by storing, indexing and replaying planning decisions. In: Drabble, B. (ed.) Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS96). pp. 158–165. Edinburgh, Scotland (May 1996)
4. Badra, F., Bendaoud, R., Bentebitel, R., Champin, P., Cojan, J., Cordier, A., Després, S., Jean-Daubias, S., Lieber, J., Meilender, T., Meilender, T., Mille, A., Nauer, E., Napoli, A., Toussaint, Y.: Taaable: Text Mining, Ontology Engineering, and Hierarchical Classification for Textual Case-Based Cooking. In: ECCBR Workshops, Workshop of the First Computer Cooking Contest. pp. 219–228. Springer, Heidelberg (2008)

5. Banko, M., Brill, E.: Mitigating the paucity-of-data problem: Exploring the effect of training corpus size on classifier. In: Proceedings of the first international conference on Human language technology research. Association for Computational Linguistics, Morristown (2001)
6. Brill, E.: Transformation-Based Learning. Ph.D. thesis, Univ. of Pennsylvania (1993)
7. Carpineto, C., Romamo, G.: Order-Theoretical Ranking. Journal of the American Society for Information Science 51(7), 587–601 (2000)
8. Cordier, A., Lieber, J., Molli, P., Nauer, E., Skaf-Molli, H., Toussaint, Y.: Wiki-Taaable: A semantic wiki as a blackboard for a textual case-based reasoning system. In: 4th Workshop on Semantic Wikis (SemWiki2009), 6th European Semantic Web Conference. pp. 88–101. Heraklion (2009)
9. Díaz-Agudo, B., Gerváz, P., González-Calero, P.A.: Adaptation Guided Retrieval Based on Formal Concept Analysis. In: Case-Based Reasoning Research and Development: Proceedings of the Fifth International Conference on Case-Based Reasoning, ICCBR-03. pp. 131–145. Lecture Notes in Artificial Intelligence 2689, Springer (2003)
10. Díaz-Agudo, B., González-Calero, P.A.: Classification Based Retrieval Using Formal Concept Analysis. In: Aha, D.W., Watson, I. (eds.) Case-Based Reasoning Research and Development — Fourth International Conference on Case-Based Reasoning (ICCBR-01). pp. 173–188. Lecture Notes in Artificial Intelligence 2080 (2001)
11. Ganter, B., Wille, R.: Formal Concept Analysis. Springer, Heidelberg (1999)
12. Gervás, P., Hervás, R., Recio-García, J.: The Role of Natural Language Generation During Adaptation in Textual CBR. In: 4th Workshop on Textual Case-Based Reasoning: Beyond Retrieval, ICCBR07 (2007)
13. Ihrig, L., Kambhampati, S.: Derivation replay for partial-order planning. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94). pp. 992–997 (1994)
14. Lamontagne, L., Bentebibel, R., Miry, E., Despres, S.: Finding Lexical Relationships for the Reuse of Investigation Reports. In: 4th Workshop on Textual Case-Based Reasoning: Beyond Retrieval, ICCBR07 (2007)
15. Lamontagne, L., Lapalme, G.: Textual reuse for email response. In: Advances in Case-Based Reasoning. pp. 234–246. Springer, Heidelberg (2004)
16. Marcus, M., Santorini, B., Marcinkiewicz, M.: Building a large annotated corpus of English: The Penn Treebank. Computational linguistics 19(2), 313–330 (1994)
17. Messai, N., Devignes, M.D., Napoli, A., Smaïl-Tabbone, M.: Querying a Bioinformatic Data Sources Registry with Concept Lattices. In: Dau, F., Mugnier, M.-L. aud Stumme, G. (eds.) ICCS. pp. 323–336. Springer, Heidelberg (2005)
18. Smyth, B., Keane, M.T.: Using adaptation knowledge to retrieve and adapt design cases. Knowledge-Based Systems 9(2), 127–135 (1996)
19. Veloso, M.: Planning and Learning by Analogical Reasoning, LNAI, vol. 886. Springer, Heidelberg (1994)