Machine Learning Approach to Automated Correction of LaTeX Documents

Kirill Chuvilin
Institute of Computing for Physics and Technology, Protvino, Russia
Moscow Institute of Physics and Technology (State University), Moscow, Russia
kirill.chuvilin@gmail.com

***Abstract*—The problem is the automatic synthesis of formal correcting rules for LaTeX documents. Each document is represented as a syntax tree. Tree node mappings of initial documents to edited documents form the training set, which is used to generate the rules. Rules with a simple structure, which implement removal, insertion or replacing operations of single node and use linear sequence of nodes to select a position are synthesized primarily. The constructed rules are grouped based on the positions of applicability and quality. The rules that use tree-like structure of nodes to select the position are studied. The changes in the quality of the rules during the sequential increase of the training document set are analyzed.**

I. Introduction

The reason for this work is the preparation of conference proceedings in 2007–2012. Many scientific conferences and publishing houses get source texts from authors in the LaTeX format. Each publisher has certain traditions and requirements for the material to publish. These rules include the design of headings, lists, tables, bibliography, equations, numbers, etc. The errors associated with non-compliance with these rules are called *typographical* errors. The sources sent by authors usually contain a significant amount (dozens per page) of such errors, the correction of which is done manually by editors. There are tools to facilitate manual proofreading [1], but nevertheless the processing of a single page takes up to two hours of time. Therefore, an automation of typographical errors correcting process is actual to reduce the time and volume of manual work.

Generally speaking, the idea to automate the correction of texts is not new [2], and at the moment there are some qualitative tools for automatical search and fixing spelling errors using dictionaries and word forms morphological analysis [3]. In addition, a similar problem arises for intelligent error correction in search queries using lexical and statistical characteristics [4]. But these approaches are not applicable for the correction of typographical errors that are discussed in this article and are associated not only with the textual content of the document, but also with formatting layout. Often the local information in the text is not sufficient to describe an error, but also the knowledge of the context (additional information about the position in the document structure) is required.

On the other hand, there is a research field devoted to the improvement of programs source code characteristics (the probability of errors in individual modules, the degree of connectivity modules, etc.). Known methods allow to measure the characteristics with the analysis of repositories change history and then use them to find errors in the code [5], [6]. They allow to create recommendation systems to improve the quality of program source code [7]. Documents in the LaTeX format can be regarded as the source code, which is used by the TeX compiler, but it is not common to use the suitable for further analysis repositories in publishing practice. There are no uniform standards. And in addition, the text contents of documents cannot be subject for such a processing.

Thus there is a need for a new research aimed directly at the automation of typographical error correction. The proposed approach is the following. Editors works with the system, which itself determines the possible fix positions in the source text and offers him a replacement. If they agree with the replacement, the only thing that is needed is to press the appropriate button. If they don't agree, then they make editing manually.

Correcting rules can be defined manually on the basis of practical experience of editors. However, due to the significant number and diversity of situations and behaviours this will rather increase the amount of work, especially at the initial stage. Therefore, we propose to build the rules using the already processed set of document pairs. A document that hasn't been proofread will be called a *draft copy*, and a proofread document will be called a *clean copy*.

The problem of document transformation rules automatic synthesis using the training set composed of "draft–clean copies" pairs is considered in [8]. However, the rules obtained with such a way have several drawbacks including the incomplete coverage of the error positions in the non-training document set and the wrong suggested replacement.

This article deals with two approaches to improve the quality of a rule set using the structure complexity.

II. Identification of differences between documents

LaTeX documents in the text representation are sequences (can be nested) of *lexemes* of the following types: syntactic brackets ({ and }), space, horizontal padding, vertical padding, paragraphs separator, line break, symbol, number, word letter, command, tag, wrapper (tag with the closure), equation, superscript or subscript, label, linear dimension, path to a file or a folder, list, list item, floating box, image, mathematical binary operator, mathematical postoperator, mathematical preoperator, table, table cell separator, table settings, not processing (raw) data.

In addition, such files have a natural tree structure (*syntax tree*) [8], by exploring which you can get all the necessary

information to describe a proofreading fix. Nodes of such structures will be called *tokens*. The following types of tokens are taken into account to construct a syntax tree: symbol (letter, digit, mathematical operation, quote sign, dash, etc.), command, command parameter, environment, environment body, space, paragraph separator, word, number, label, linear dimension, path to a file, tabular parameters, not processing fragment (for example, the text inside a `verbatim` environment). Each token can be matched by one of the lexeme types. Examples for token of each type are in Table I.

TABLE I. EXAMPLES OF TOKENS

Token type	LaTeX source sample
Symbol	`height 1.2\, m`
LaTeX command	`\includegraphics [width=10cm] {../figure.eps}`
LaTeX command parameter	`\includegraphics [width=10cm] {../figure.eps}`
LaTeX environment	`\begin{tabular}{c|c} height & 1.2\,m \end{tabular}`
LaTeX environment body	`\begin{tabular}{c|c} height & 1.2\,m \end{tabular}`
Space	`height`□`1.2\,m`
Paragraphs separator	Paragraph □ New paragraph
Word	`height 1.2\,m`
Number	`height 1.2\,m`
Label	`\ref{equation1}`
Linear dimension	`\textwidth=10cm`
Path to a file	`\includegraphics [width=10cm] {../figure.eps}`
Tabular parameters	`\begin{tabular}{c|c} height & 1.2\,m \end{tabular}`
Not processing fragment	`\verb|complex source|`

A syntax tree bijectively (according to the TeX compiler) defines a LaTeX document. It is convenient to formulate the correcting rules for the trees.

Syntax trees of draft and clean copies will be called *draft* and *clean trees* respectively.

The differences between the draft and the clean trees for each document pair from the training set are identified before the search of the regularities in the changes that were done by editors. Zhang-Shasha algorithm is used for this purpose [9], [10]. It implies that the following steps are allowed to be consistently applied to the syntax tree: a token removal (all child tokens go to the parent), a new token insertion to an arbitrary place, a token change. The algorithm allows to calculate the edit distance between two trees and in addition to determine which procedure should be applied to each node for the implementation of that distance. The result of its work consists of the pairs (source and target) of the not modified tokens, the pairs (source and target) of the changed tokens, the set of the removed tokens, the set of the inserted tokens.

III. RULES WITH A SIMPLE STRUCTURE

The initial set of rules is built after obtaining the mapping between the draft and clean trees [8]. Each synthesized rule characterized by the *pattern* (the sequence of adjacent tokens with the same parent), the *localizer* (the token whose child tokens the pattern is applied to) and the *action* (the operation aimed at changing the syntax tree). Thus, a rule allows to locate the error (to determine its position in the syntax tree and consequently in the original document text; it is considered that the error is in the piece of the text that corresponds to the rule pattern tokens), and it offers the option of correcting (the change of the syntax tree and hence the initial document text) on the basis of the rule action.

Definition 1: Left pattern chain of radius r is a sequence of not more than r adjacent tokens with the same parent. *The left chain start* is its most right token.

Right pattern chain of radius r is a sequence of not more than r adjacent tokens with the same parent. *The right chain start* is its most left token.

Let token x of a draft tree be removed or changed to token y. Then the localizer is the parent token of x, the pattern is composed of the left and right closest to x pattern chains and of token x. In such cases, token x will be called *the target token of the rule*. The rule action is to remove the target token or to change it to a token y, depending on the type of the rule.

Let token y be added to a clean tree. Then the localizer is the prototype of the parent token (if it exists) of y, the pattern is composed of the left pattern chain that starts from the prototype of the y's left neighbour (if it exists) and the similar right pattern chain. The rule action is to insert the token y between the left and the right pattern chains.

Example 1 (A token insert rule): An extra space is often inserted between two standing next equations to improve the readability of the text. The mapping between syntax trees, which appears while changing `If $S$, $Q$, then` to `If $S$,\; $Q$, then` is shown in Fig. 1.

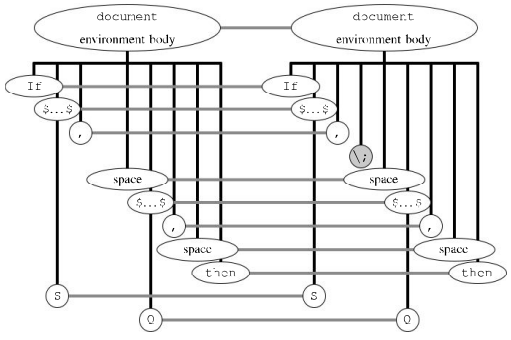

Fig. 1. A token insert rule example.
`If $S$, $Q$, then` → `If $S$,\; $Q$, then`

The meaning of change is the insertion of a `\;` symbol between the comma and the space. But an extra space is

not always needed between a comma and a space, even if there is an equation before the comma or an equation after the space. The correct condition is the presence of equations before the comma and after the space.

Therefore in this case the left pattern chain consists of the comma token and the equation token, the right pattern chain consists of the space token and the equation token, the pattern consists of the equation token, the comma token, the space token and the equation token. The rule action is to insert a `\;` symbol token between the comma token and the space token.

Example 2 (Token change rules): One of the most usual typographical errors is to use the "..." quotes instead of <<...>> quotes in Russian-language sources. Let the mapped parts of draft and clean trees look as shown in Fig. 2, this corresponds the mapping of `''word` to `<<word` in a `document` environment.

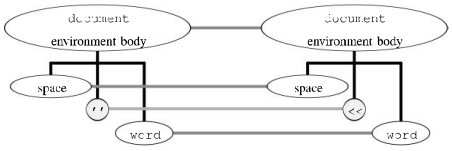

Fig. 2. A token change rule example.

`''word` → `<<word`

In this case the localizer is the `document` environment body token, the left pattern chain consists of the space token, the right pattern chain consists of the word token, the pattern consists of the space token, the `''` symbol token and the word token. The rule action is to change the token that is between the space token and the word token to a `<<` symbol token.

The similar rule is built to change right quotes.

On the other hand, some of the rules are not applicable to all the document fragments. This confirms the need to take a context into account for the creation and use of the rules. For example, a number of pages in a list of references is usually denoted by "Pp" for English-language sources. But it is usual to see a single letter "P" designation in practice. So there is a need to change "P" to "Pp", but such a modification is likely to be incorrect for a random fragment of the document. Therefore, this rule is built only for the `thebibliography` environment body.

The mapping between syntax trees, which appears while changing `, P` to `, Pp` is shown in Fig. 3.

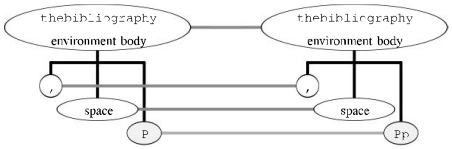

Fig. 3. A token change rule example.

`, P` → `, Pp`

In this case the localizer is the `thebibliography` environment body token, the left pattern chain consists of the comma token and the space token, the right pattern chain is empty, the pattern consists of the comma token, the space token and the "P" word token. The rule action is to change the pattern last token to a "Pp" word token.

A. Searching of matches to rules

A token l is considered to correspond to a rule localizer if their types and their lexeme types are matched.

A continuous sequence of l child tokens is sought according the following principles:

- token types and lexeme types must be matched for all the tokens of the pattern chains and the corresponding child tokens of l,
- the target token must be full matched to the corresponding child token of l.

Definition 2: Rule position in the syntax tree is a union of the token that corresponds to the rule localizer and the set of tokens matching to the pattern. *Rule generating position* is the position that corresponds the item of the mapping between syntax trees that was a basis for the rule. *Rule position set* in a document set is the set of all the positions in the syntax trees of these documents that match to the rule.

B. Preliminary estimation of a rule

Preliminary quality estimations are calculated for each rule using the training set [11]. Let d_t and c_t be the sizes of the rule position sets in the sets of draft and clean copies respectively.

Definition 3: Preliminary (in the training set) precision of a rule is the ratio of the rule position matching to the draft copies only set size to the total number of the found positions:

$$\frac{d_t - c_t}{d_t}.$$

C. Selection of optimal patterns

The tokens set that forms a rule pattern can be defined in different ways. According to the experiment results, we can conclude that the maximum patterns do not always give the best result [8]. In this article, the optimal pattern is selected by the following criteria:

1) the rule preliminary precision must not be less than 0.9;
2) the pattern with the smallest size is sought among all the patterns that provide the selected precision;
3) the rule with the greatest precision is sought among all the rules with the selected pattern size.

D. Reduction of a rule set

It turns out that rules are redundant after processing all the differences among the training set trees as many of them are duplicated.

To eliminate this effect, the reduction process is used, which means the removing of some rules, so that the general set of the identified regularities stays the same.

Definition 4: A rule A *absorbs* a rule B, if their actions are the same and the rule B position set in the draft copies of the training set is a subset of the rule A position set.

Rules that can be absorbed by the other rules are incrementally removed from a generated rules set.

Definition 5: Rule generating position set is the set of generating positions of this rule and all the rules absorbed by this rule.

IV. Quality estimations

The experiment was performed to estimate the quality of a rule set. It used 85 pairs of draft and clean copies of articles from the IIP-8 conference. The adaptive learning of the rule set was simulated. For this purpose the training set of the document pairs used for rule synthesis increased gradually: 2, 3, 4, 6, 9, 13, 19, 28, 42, 63. Let $S_1 \subset \ldots \subset S_{10}$ be the generated ten training sets of the document pairs, S_{11} is the set of all the document pairs. The control set was formed of the document pairs added to the training set at the next step: $S_{i+1} \setminus S_i,\ i = 1, \ldots, 10$.

Quality estimates of the synthesized rules and the rule sets were calculated [11]. The sets sequences were constructed for 50 times, the data for all the tests was averaged.

Let for any rule p_t be the rule generating position set size, d_c and c_c be the rule position set sizes in the sets of draft and clean copies respectively of the control document set, p_c be the rule position set size in the draft copies set that corresponds the correct changes of the control document set.

Definition 6: Adjusted preliminary precision of a rule is the ratio of the rule position really matching to the draft copies only set size to the total number of the found positions:

$$\frac{p_t}{d_t}.$$

Definition 7: Control precision of a rule is the ratio of the rule position in the control document set that match to the draft copies only set size to the total number of the found positions:

$$\frac{d_c - c_c}{d_c}.$$

Definition 8: Adjusted control precision of a rule is the ratio of the rule position in the control document set that match to the correct changes set size to the total number of the found positions:

$$\frac{p_c}{d_c}.$$

It is possible to determine the quality evaluations of a rule set at this point.

Definition 9: Let $P(A_1), \ldots, P(A_k)$ be the preliminary precisions of rules $A_1, \ldots, A_k$ respectively, and all these rules have the same target token. *A weight of a rule* A_i is

$$W(A_i) = \frac{P(A_i)}{\sum_{j=1}^{k} P(A_j)}.$$

Definition 10: Let $E(A_i)$ be a number equal to 0 if the rule A_i corresponds to a correct fix and 1 otherwise. Then the expression

$$e_x = \sum_{i=1}^{k} W(A_i)E(A_i) = \frac{\sum_{i=1}^{k} P(A_i)E(A_i)}{\sum_{i=1}^{k} P(A_i)}$$

defines *the average error of a rule set* at the selected token x.

Let E_t and E_c be the average errors sums of a rule set at all the tokens of the draft trees in the training and the control document sets respectively, N_t and N_c are the different position numbers of a rule set in the sets of draft copies of the training and the control document sets respectively, D_t and D_c are the editing distances sums for all the draft and clean tree pairs of the training and the control document sets respectively.

Since the rules are generated while tokens insertion, deletion or changing only, and the sum of such operations is the edition distance, the following definitions are correct [11].

Definition 11: Preliminary (in the training document set) precision of a rule set is

$$\frac{N_t - E_t}{N_t}.$$

Control (in the control document set) precision of a rule set is

$$\frac{N_c - E_c}{N_c}.$$

Definition 12: Preliminary (in the training document set) recall of a rule set is

$$\frac{N_t - E_t}{D_t}.$$

Control (in the control document set) recall of a rule set is

$$\frac{N_c - E_c}{D_c}.$$

As can be seen from Fig. 4, the reduction can significantly reduce the number of rules. This means that most of the differences between the draft and the clean copies are typical for the whole document set.

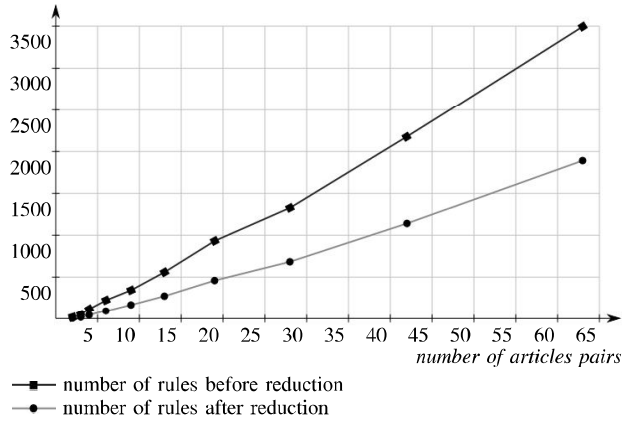

Fig. 4. The numbers of the generated rules.

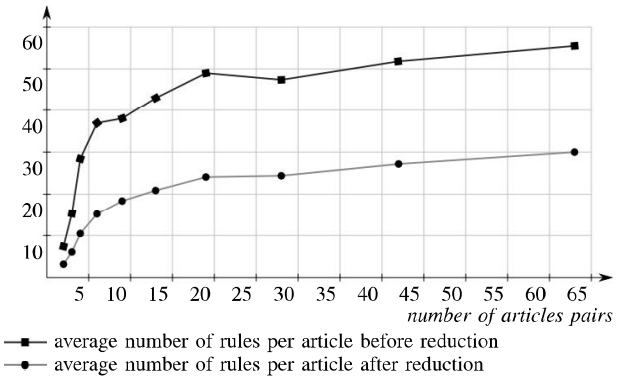

Fig. 5. The average numbers of rules for one "draft-clean" pair.

The graphs in Fig. 5 correspond to the increasing functions. From this we can conclude that there's not a negligible number of unique differences between draft and clean copies.

The curves in Fig. 6 that correspond to the adjusted preci-
sion ne
can to
the] ns
that od
genc

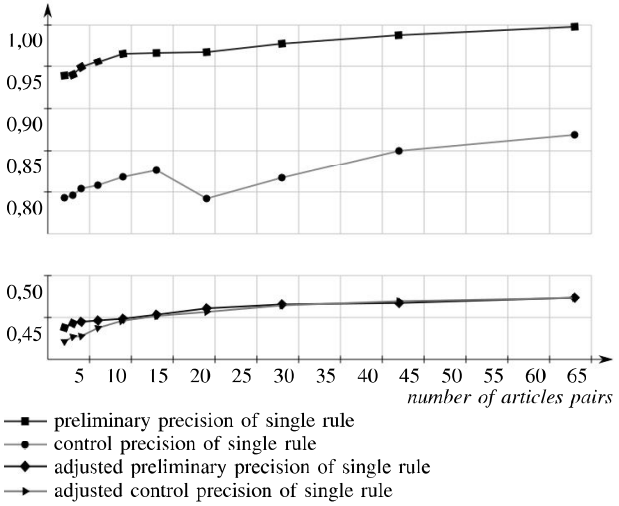

Fig. 6. The average quality estimates of a single rule.

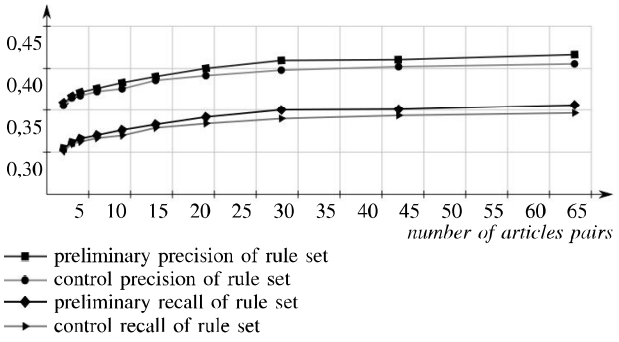

Fig. 7. Estimates for the precision and the recall of the rules with the simple structure set.

On the other hand, the precision and the recall of the rule sets don't exceed 50%. For precision this means that there are different rules with similar patterns. Lack of recall can be explained by the fact that the types of rules discussed earlier are not enough to describe editor operations.

V. Group rules

In practice, there are cases where an editor changes, deletes or inserts more than one token. For example, the moving of the token to another position is a union of removing and inserting the token. To increase the range of the recognizable edits we will use the rule grouping.

Example 3 (Group rules): Publishing design standards may vary dashes. This leads, for example, to the need to change `~---` to `''---`.

The mapping between syntax trees, which appears while changing `def~--- definition` to `def ''--- definition` is shown in Fig. 8.

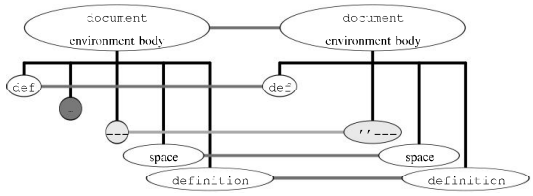

Fig. 8. A group rule example.
`def~--- definition` → `def ''--- definition`

The first fragment consists of the non-breaking space (`~`) token and the dash symbol (`---`) token, the second fragment consists of the inseparable dash symbol (`''---`) token. Each rule with a simple structure can change only one token, that is why such a replacement cannot be implemented. On the other hand, the deletion of the non-breaking space token standing in front of the dash token and the change of the dash token to the inseparable dash token together give the desired replacement.

The similar problem arises correcting one of the most usual typographical errors when a hyphen is used instead of a dash in Russian-language sources. The corresponding mapping between syntax trees is shown in Fig. 9.

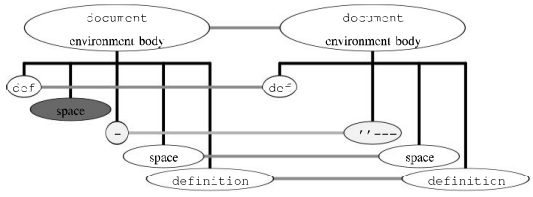

Fig. 9. A group rule example.
`def - definition` → `def ''--- definition`

Another similar example is the proper use of a hyphen character. The mapping between syntax trees, which

appears while changing `$\sigma$ - covering` to `$\sigma$''=covering` is shown in Fig. 10.

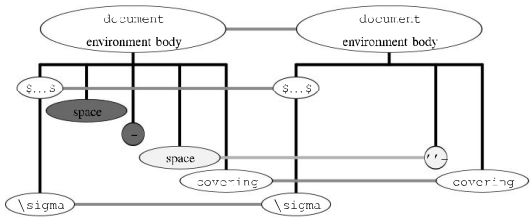

Fig. 10. A group rule example.
`$\sigma$ - covering` → `$\sigma$''=covering`

An insertion of more than one token is necessary sometimes. The mapping between syntax trees, which appears while changing `n.k.f.` to `n.\;k.\;f.` is shown in Fig. 11. Two tokens of symbol `\;` are added in this case.

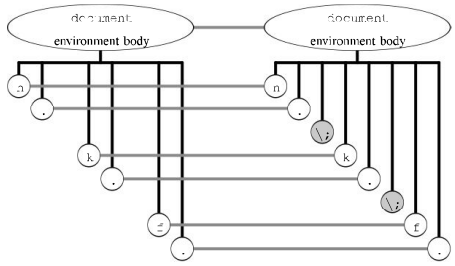

Fig. 11. A group rule example.
`n.k.f.` → `n.\;k.\;f.`

Suppose that there are two rules with such positions that:

- the tokens corresponding to the localizers are the same;
- the sets of the tokens corresponding to the patterns have mutual elements.

Then we construct a new *group rule* the localizer of which is the same as the localizer of the given rules and the pattern is formed by the union of their patterns. The built rule is added to the rule set, if its preliminary precision is higher than the preliminary precisions of both old rules.

It is considered that the position in a syntax tree of a document matches to the group rule if it matches to each of the grouped rules. This takes into account the relative position of their patterns: overlap should occur by the same tokens, which it happened at the time of the group rule generation.

Fig. 12 shows the quality estimations of the rule set using group rules built in accordance with the experiment described earlier. It can be seen that this approach yielded the precision significantly more than a half, but the recall is still at the level of 50%.

VI. Rules with tree patterns

Another way to improve the precision and the recall of a rule set is the complication of a pattern structure. A pattern

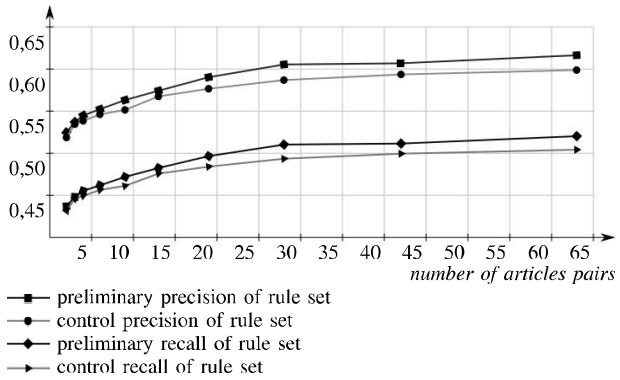

Fig. 12. Estimates for the precision and the recall of the rule set taking into account the group rules.

of a simple structure rule allows to use the adjacent tokens only to determine the position. Generally speaking, it doesn't mean the use of all the text corresponding to these tokens, because it doesn't take into account the structure and the contents of the subtrees, which roots form the pattern.

A pattern of a *tree-like rule* is constructed of two *pattern trees: left and right*. In this case *the length of the pattern* is the number of tokens in these trees.

The generation of such rules, the selection of the optimal patterns and the reduction are similar to the generation of rules with a simple structure.

Let token x of a draft tree be removed or changed to token y. Then the localizer is the parent token of x, the pattern is composed of the left and the right pattern trees such that all the child tokens of the left pattern tree root form a left pattern chain closest to x, and the child tokens of the right pattern tree root form a similar right pattern chain. In such cases, token x will be called *the target token of the rule*. The rule action is to remove the target token or to change it to a token y, depending on the type of the rule.

Let token y be added to a clean tree. Then the localizer is the prototype of the parent token (if it exists) of y, the pattern is composed of the left pattern tree such that the child tokens of its root forms a left pattern chain that starts from the prototype of the y's left neighbour (if it exists) and the similar right pattern tree. The rule action is to insert the token y between the left and the right pattern chains.

The following steps are used to select the optimal pattern:

1) the rule preliminary precision must not be less than 0.9;
2) the pattern with the smallest sum size of the left and the right pattern trees is sought among all the patterns that provide the selected precision;
3) the rule with the greatest precision is sought among all the rules with the selected pattern size.

The searching of match positions is done with the following way. A token l is considered to correspond to a rule localizer if their types and their lexeme types are matched. The patter trees and all their subtrees are tested at each level started from the token l child tokens according the following principles:

- the most right child tokens of the left subtrees must be equal to the corresponding pattern tokens,
- the most left child tokens of the right subtrees must be equal to the corresponding pattern tokens,
- each subtree must match to the corresponding child token.

Example 4 (Rules with tree patterns): Another example of the most usual typographical errors is the inclusion of a punctuation in an equation body. Let the mapped parts of draft and clean trees look as shown in Fig. 13, this corresponds the mapping of `$ ,$` to `$ $,` in a `document` environment.

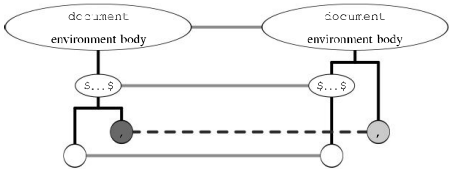

Fig. 13. A tree-like rule example.

`$ ,$` → `$ $,`

Such a replacement can be described as a union of a comma token insertion after the equation token that contains comma at its end and the comma token that is after the equation token deletion. But the inside equation tokens are required to determine whether a comma is in the equation. Thus a simple structure rule pattern is not enough.

So the rule of a comma addition is described by a tree pattern, the left pattern subtree consists of the equation token containing the comma token the right pattern subtree is empty. The action is to insert a comma token after the left subtree.

Tree-like rules can also be used to form group rules. The mapping between syntax trees, which appears while changing `$\textbf{E}|S|$` to `$\Expect|S|$` is shown in Fig. 14. The corresponding rule is a tree-like rule because not only the `\textbf` command token is required but also its parameter is needed. For example `\textbf{D}` must be replaced with `\Var` but not with `\Expect`. The rule is a group rule because not only the `\textbf` command token must be changed but also its parameter token must be deleted.

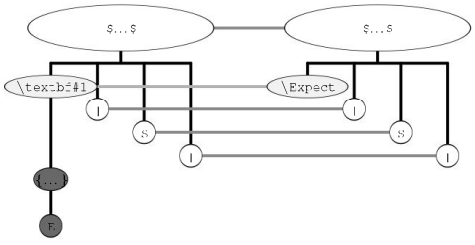

Fig. 14. A tree-like group rule example.

`$\textbf{E}|S|$` → `$\Expect|S|$`

Fig. 15 shows the quality estimations of the rule set using tree-like rules built in accordance with the experiment described earlier. It can be seen that this approach can significantly increase the precision of the generated rule set.

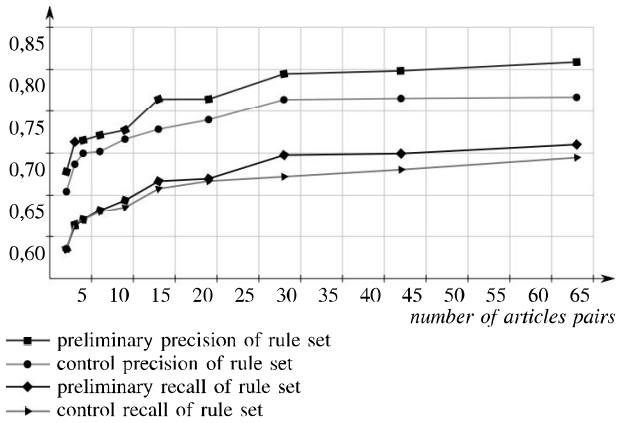

Fig. 15. Estimates for the precision and the recall of the rule set taking into account the tree-like rules.

It is important to note that the resulting recall is greater than 70%, i.e. less than 30% is not recognized, and the precision is greater than 75%, this allows to use the generated rule set on the practice of semi-automatic error correction.

VII. Conclusion

Rules with a simple structure described by a linear pattern are examined. The method for constructing such rules on the basis of the training sample composed of "draft-clean" pairs and the use of them is described. The optimal patterns selection and the reduction processes are shown.

The precision and recall of rule sets and the precision for single rule estimations method is proposed. An experiment was performed using the articles of the IIP-8 conference. The dependence of the precision and recall of a rule set on the number of items that are used for training was studied.

The experiment shows that the precision and the recall of rules with a simple structure are not high. This is justified by a small number of supported operations (change of only one token at a time is handled) and an excessive simplicity of the patterns (it is required to have an opportunity to explore more than one child level of the localizer token).

Two approaches were proposed to improve the rule quality: the construction of group rules and rules with a tree pattern. The rule grouping allows operations with a set of tokens at the same time. Tree patterns allow access to different layers of the document syntax tree.

The dependence of the precision and recall of a rule set on the number of items that are used for training was studied again for group and tree-like rules under the same conditions as for the rules with a simple structure. Each approach allows to significantly improve the precision and the recall of the generated rule sets.

The results can be considered as acceptable for a practical use.

ACKNOWLEDGMENT

This work was supported by the RFBR grants: 16-37-60049, 16-07-01267.

The author is deeply indebted to Prof. Konstantin Vorontsov for the wise scientific supervision in the field of machine learning and the statement of this research original problem. Also I am grateful to Alexey Rozanov (the head of the publishing house "Fizmatkniga") for the extensive proofreading experience with LaTeX. And in addition I would like to thank Tatiana Alenkina for the help with the correct formulation in English of this article ideas and results.

REFERENCES

[1] J. André and H. Richy, "Paper-less editing and proofreading of electronic documents.", 1999. Web: http://www.irisa.fr/imadoc/articles/1999/heidelberg.pdf.

[2] I. A. Bol'shakov, "Problems of automatic text correction in inflected languages" [Problemy avtomaticheskoj korrekcii tekstov na flektivnyh yazykah], *Results of science and technics. Ser. Probability theory. Math statistics. Theory of cybernatics.,*, vol.28, M.: VINITI. 1988, pp. 111–139.

[3] "Lightproof grammar checker development framework", 2013. Web: http://extensions.services.openoffice.org/project/lightproof

[4] M. F. Panina, A. V. Baitin, I. E. Galinskaya, "Context-independent autocorrection of query spelling errors" [Avtomaticheskoe ispravlenie opechatok v poiskovyh zaprosah bez ucheta konteksta], *Computer linguistics and intelligent technologies. On matherials of annual international conference "Dialog"*, issue 12, vol. 1, 2013, pp. 556–567.

[5] C. Williams, J. Hollingsworth, "Automatic Mining of Source Code Repositories to Improve Bug Finding Techniques", *IEEE Transactions on Software Engineering table of contents archive*, vol.31, No.6, 2005, pp. 466–480.

[6] E. G. Knyazev, "Methods for detection of patterns of evolution code" [Metody obnaruzheniya zakonomernostej ehvolyucii programmnogo koda], *Proceedings of the XII All-Russian Scientific Conference Telematics 2007. ITMO University, Saint Petersburg, Russia.*, vol. 4, 2007, pp. 435–436.

[7] F. Madou M. Agüero., G. Esperón, D. López De Luise, "Software for Improving Source Code Quality", *World Academy of Science, Engineering and Technology*, vol.59, 2011, pp. 1259–1265.

[8] K. V. Chuvilin, "The use of syntax trees in order to automate the correction of LaTeX documents" [Ispol'zovanie sintaksicheskih derev'ev dlya avtomatizacii korrekcii dokumentov v formate LaTeX], *Computer Research and Modeling*, vol.4, No.4, 2012, pp. 871–883.

[9] K. Zhang, D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems", *SIAM Journal of Computing*, No.18, 1989, pp. 1245–1262.

[10] K. V. Chuvilin, "An efficient algorithm for LaTeX documents comparing" [Ehffektivnyj algoritm sravneniya dokumentov v formate LaTeX], *Computer Research and Modeling*, vol.7, No.2, 2015, pp. 329–345.

[11] K. V. Chuvilin, "Adaptive learning of LaTeX documents correction rules" [Adaptivnoe obuchenie pravil korrekcii dokumentov v formate LaTeX], *Report of the 9th International Conference "Intellectualization of Information Processing" IIP-2012*, pp. 652–655.