A Representation of Tree Matching and Rewriting for Model Transformation

Yuki Hirano[1,a)] Nobuhiko Ogura[1,b)]

Abstract: This study proposes a simple representation of tree processing for model transformation. System development with model transformation often processes the definition part of behavior and scope in the model. Even though models are typically represented by graphs, in such cases , tree processing is required. However, the notation of tree processing tends to be complicated because of the problem that recursion and complex conditional branching are required. To reduce such difficulty, we propose a simplified representation of tree bonding, constructing and matching for model transformation. With these attempts, tree processing can be written in short notation like processing to other data structures such as strings or lists. Also, that improves the efficiency and maintainability of model transformation.

Keywords: Tree rewriting, Model transformation, Tree searching, Tree data structure, Model-Driven Development

1. Introduction

Model transformation is effective for consistent and traceable development and may also be used in model-driven development for embedded systems. System development with model transformation often requires processing behavior and scope written in the model. There are cases where it becomes necessary to process the tree in the model transformation, even though the model is typically represented by a graph. For example, the syntax tree of the action description in the action language and the scope tree that defines the visible range of the scope are the trees described in the model. In the model transformation for these models, it is necessary to process the tree data structure. However, the notation of tree processing tends to be complicated because many tree processing requires scanning of trees using recursion to adapt complex tree structure.

Therefore, we propose a representation of the tree reconnection process for tree rewriting. Moreover, our representation provides tree construction and matching, which are required for many tree processing including tree connection. These attempts allow the processing of trees to be written short and intuitive form. We implemented the proposed method and tried that it could process trees. In the future, we will confirm that these functions are effective for other typical model processing. Moreover, we aim to provide these functions as a library to assist model transformation.

In the future, the tree processing by the proposed method can be represented by a similar notation as other data structures such as strings and lists. This attempt aims to contribute to improving the efficiency and maintainability of model transformation. Furthermore, we will study matching based on both paths and patterns, and matching patterns that specialize in extracting information. As future issues, it is necessary to study pattern representation in matching and an interface suitable for tree connection. We will also consider a method for quantitatively evaluating that the proposed method contributes to reducing the problem of model transformation.

[1] Tokyo City University Yokohama Campus, 3-3-1, Ushikubonishi, Tsuzuki-ku, Yokohama-shi, Kanagawa, Japan
[a)] g2183128@tcu.ac.jp
[b)] ogura@tcu.ac.jp

2. Tree Processing Representation for Tree Bonding

First, as preliminary of connection processing, we explain tree construction and tree matching. After that, the connection process is explained.

2.1 Subtree Generation

Tree processing such as matching and insertion requires quick construction of subtree. However, the notation of construction is complicated even for small subtree. Therefore, we propose a shorter notation that can represent tree construction. In the proposed notation, the following symbols (i) to (vii) are used. (i) >... The left side is the parent node on the right side (ii) +... Both sides are child nodes of the same parent. (iii) #... Define the value of the node attribute. (iv) {}... Specify the attribute name. (v) ()... The bound parts are preferentially combined. (vi) \... Escape the next character in the node name. (vii) ##... Specify node as "place". Fig.1 is the syntax tree of the "if" statement

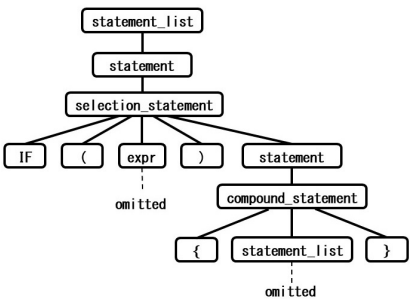

Fig. 1 Syntax tree of "if"

(partially omitted) and can be written as follows by the proposed notation.

```
statement_list > statement > selection_statement >
IF + \( +(expr > ...)+ \) + statement >
compound_statement > \{ +(statement_list > ...)+ \}
```

The role of the place represented by ## at this time is explained in Section 2.3.

2.2 Subtree Matching

Tree matching processing is required for many processes such as identifying the processing part and extracting information from the tree. However, the matching process of a tree tends to be complicated because the notation is complex compared to other data structure such as strings or lists. In addition to exact pattern matching, flexible matching may be required. For example, in the matching process, it is necessary to consider the case where the matched ranges overlap. We use an extended section 2.1 notation. In the notation, by writing a minus sign in front of a node, subtree rooted at that nodes will not be excluded from subsequent scans. (M1) in Fig.2 is an example of matching that allows duplication of matching range by specifying a node .

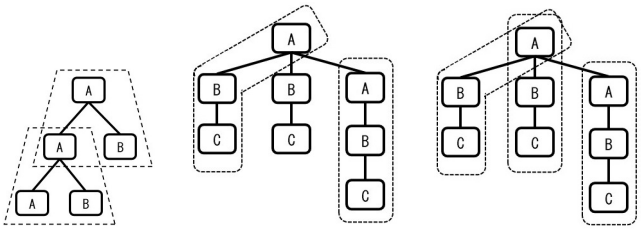

Fig. 2 Samples of tree matching

Other tree matchings may be needed. First, it may be necessary to match using ancestor paths as well as tree patterns. (M2) in Fig. 2 is an example of matching only patterns. While (M3) in Fig. 2 is an example of matching that combines paths and patterns. In the representation of (M3), the semicolon separates the path A and the pattern B >C.

Also, in matching, both the node name and other attributes of the node may be required. Matching of arbitrary nodes, repetition of patterns, etc. can also be conditions for matching. It is a future task to expand the matching notation and propose a more flexible notation.

2.3 Connection of Whole Tree and Insertion Part Tree

Connecting the whole tree and the subtree is a necessary process for rewriting the tree. As an example of the tree insertion process, the syntax tree of program 1 in Fig.3 is rewritten and changed as in program 2. T1 in Fig.4 is the syntax tree of program 1, and T2 is the syntax tree of program 2. In this rewriting example, the “if” syntax tree shown in Fig.1 is inserted in T1. In T2, as shown in the following (i) to (iii), multiple edges are reconnected. (i) e1 → e1´ The child node has been rewritten from S2 to S2´. (ii) e2 → e2´ The parent node has been rewritten from S2 to S2´. (iii) A new edge e3 is added and S2 is reconnected to the inserted “if” subtree.

In the proposed expression, when constructing a subtree, the part that connects to the whole tree can be specified as a ”place“. In the subtree construction notation proposed in Section 2.1, it is possible to specify a node as a ”place“ using the ## symbol. The ”place“ itself and the edge pointing to the joint can be easily obtained from the subtree. When connecting trees, ”place“ can be used as a joint.

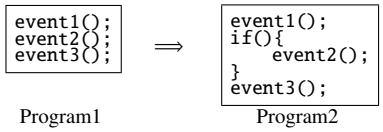

Fig. 3 Programs before and after syntax tree processing

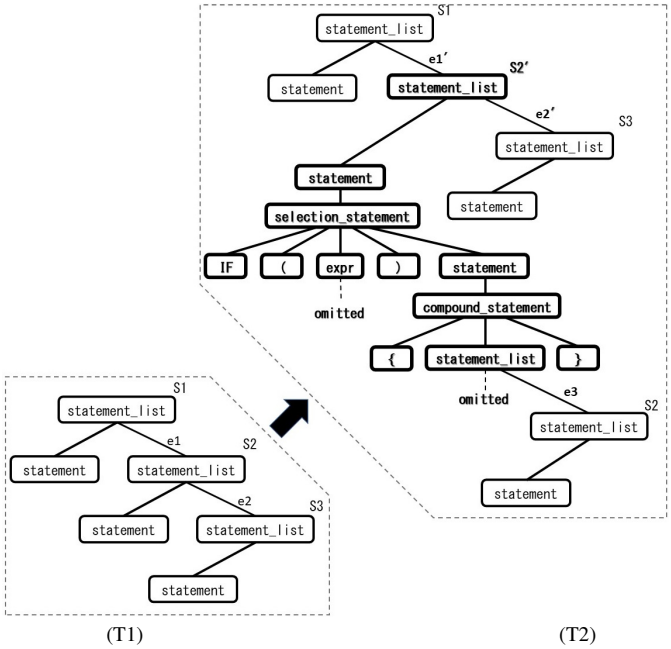

Fig. 4 Insert “if” subtree

3. Future Issues or Future Development

As a future task, it is necessary to study more flexible matching expressions. For example, matching using an amorphous tree represented like a regular expression may be suitable. In addition, in order to examine an intuitive matching pattern expression suitable for a tree structure, we refer to Emmet and yml related to data structure expression. Further, regarding the tree connection in the proposed method, the process of reconnecting multiple edges to the corresponding nodes is still complicated. It is necessary to extend the proposed method and consider an interface that can describe multiple reconnection processes at once. As a future plan, we will examine the validity of the processing representation in a typical model transformation example, and support multiple tree-structured data representations and languages.

4. Related Works

As a related study, there is a study of graph rewriting by Barendregt et al[1]. Studies on term graph rewriting examine in detail the theoretical aspects of tree rewriting. In addition, there are Emmet and YAML formats that handle tree structures. These formats provide simple tree construction notations for different purposes.

5. Conclusion

In this study, we examined a concise expression method for basic processing such as connection, matching, and generation, which is required for tree structure processing for model transformation. In addition, these functions were implemented using the programming language Perl, and future issues in tree structure processing in model transformation were clarified.

References

[1] Barendregt H.P., van Eekelen M.C.J.D., Glauert J.R.W., Kennaway J.R., Plasmeijer M.J., Sleep M.R. (1987) Term graph rewriting. In: de Bakker J.W., Nijman A.J., Treleaven P.C. (eds) PARLE Parallel Architectures and Languages Europe. PARLE 1987. Lecture Notes in Computer Science, vol 259 ,pp.141–158. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-17945-3_8