Pragmatic Graph Rewriting Modifications

P.J. Rodgers and N. Vidal
Computing Laboratory, University of Kent, U.K. email: P.J.Rodgers@ukc.ac.uk, N.Vidal@ukc.ac.uk

Abstract

We present new pragmatic constructs for easing programming in visual graph rewriting programming languages. The first is a modification to the rewriting process for nodes the host graph, where nodes specified as 'Once Only' in the LHS of a rewrite match at most once with a corresponding node in the host graph. This reduces the previously common use of tags to indicate the progress of matching in the graph. The second modification controls the application of LHS graphs, where those specified as 'Single Match' are tested against the host graph one time only. This reduces the need for control flags to indicate the progress of execution of transformations.

1. Introduction

Graph rewriting is an increasingly popular visual paradigm for applications where graphs are the dominant data structure. We discuss some on going modifications to such a language in the light of our experience in programming with the Grrr [1] graph drawing variant of the Spider programming language [2].

Spider was always envisaged as a prototype, modifiable experiment in serial deterministic programming with graph rewrites. Hence it has minimal features, which simplifies its semantics, but which can mean that programs are difficult to construct. There was a trade off between complexity a programming language, and it ease of use. As needs for changes to the paradigm become apparent, alterations to the implementation of the system are feasible because of its prototype nature. The constructs discussed here are part of an on going pragmatic process to improve the ease of use of Spider.

The two methods of modifying the rewriting method that are given in this paper are both specified in graph transformations, which are lists of graph rewrites. A graph rewrite is a pair of graphs, a LHS and a RHS. The differences between the LHS and RHS define the changes to be made to the host graph if the LHS graph of the rewrite matches in the host graph. Transformations are initiated by trigger nodes in the host graph, and the rewrites are normally tested in turn in a top down manner until one LHS matches.

The first modifier is an extra possible feature of nodes in the LHS of a rewrite. When specified by a programmer as 'Once Only' a node in a LHS of a rewrite can match with each node in the host graph at most once.

The second modifier is a feature of rewrites. A rewrite specified as 'Single Match' will only match once during the execution of that transformation. Usually, the top down matching method used by Grrr forces each LHS graph to be considered in turn as a potential match in the host graph. However, once the LHS of a Single Match Rewrite has been found in the host graph, it will not be tested for a match again.

The two new constructs are additions to the language, so we maintain backwards compatibility with previous versions of Spider.

2. The rewrite modifiers

The scope for both the new features are the trigger node in the host graph that initiated the transformation. This means that a Once Only Node that matches a node in the host graph as a consequence of one trigger node will only affect the matching initiated by the same trigger node, but has no effect on the matching of another trigger node of the same name, so that the node in the host graph can match again.

The addition of Once Only nodes is designed to reduce the use of tags in transformations. The use of tags is common when programming in Grrr as a they are used to indicate which nodes have been visited. Typically a rewrite will test for the negative presence of a tag attached to a data node, and if not present the rewrite will perform some action on the node and create a tag attached to it. The next application of the transformation will then not match with that data node. After all the relevant nodes have been matched the tags must be removed by a garbage collection transformation.

The Once Only construct allows tags to be avoided by iterating through a graph one node at a time with the node specified in the LHS as a Once Only Node. As a consequence, transformations are clearer and easier to specify, with a reduced number of primitives, including less use of negatives. Also, during execution programming steps are avoided and the host graph has more clarity because it no longer contains these tags.

The use of Once Only Nodes can be seen in Figure 2.1, which shows a Once Only Node, ‘C’, with a shaded background, in the first three LHS graphs of the transformation. This transformation is part of a program to layout a B-Tree. ‘DoBase’ places the nodes at the bottom of the tree in a line.

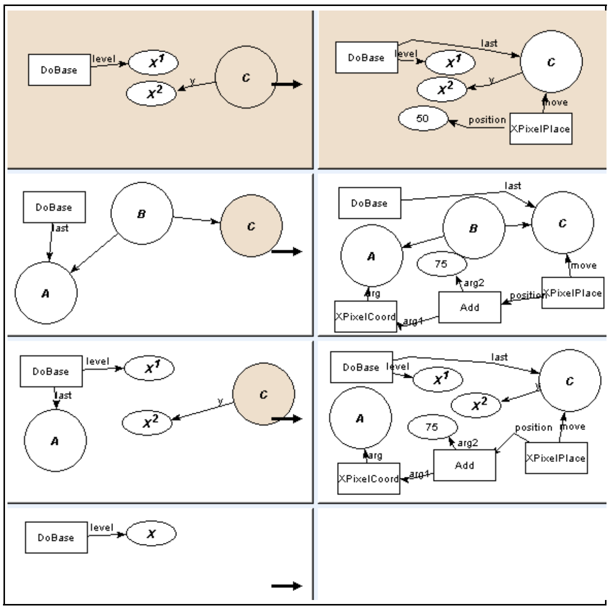

Figure 2.1: The transformation ‘DoBase’

The second new construct, Single Match, allows the control of execution of rewrites to be specified more precisely. The technique previously was to introduce flags, and as with tags, negatives in LHS graphs would prevent the subsequent matching of rewrites which created the flags. Now rewrites specified as Single Match will match a single time in the host graph, and will be ignored during subsequent applications of the trigger node. As with Once Only the overall effect is to streamline transformations, lessening the number of negatives and reducing the clutter in the host graph.

Figure 2.1 illustrates the use of Single Match Rewrites. The first rewrite is shaded, indicating that the programmer has specified that it is single match. It will only match a single time in the host graph for each instance of the trigger node initiating the transformation, so that after rewriting the host graph the rewrites can be disregarded from further matching. This also demonstrates that Single Match Rewrites may contain Once Only Nodes. There is little interaction between the two new techniques. The first transformation will be used at most a single time, and the node in the host graph that matches with ‘C’ cannot match with a Once Only Node in another rewrite when the same trigger node initiates this transformation.

An addition to the rules governing which trigger node of two in the host graph is applied first has been made. This is because of an history of matching in the host graph is now required to be stored, as all host graphs now have to have knowledge about which nodes have already been matched. Previously where two nodes were inseparable by connecting subgraphs, node label or age, either could be executed safely, as either node will have the same effect. Now if one trigger has matched a Once Only Node or Single Match Rewrite, the effects may not be the same, hence the additional rule that the trigger node furthest through its rewriting process is executed first. This retains the serial deterministic application of trigger nodes.

3. Conclusions

We have presented two new serial graph rewriting modifiers. The Once Only Nodes described allow programmers to more easily iterate through the nodes in a host graph, whilst the Single Match Rewrites allow closer control of execution order. The net effect is to make programming in Grrr simpler and more effective.

We know of no obvious analogous constructs in other systems, either visual or textual. This confirms our belief in the interesting and unique nature of the group of programming paradigms inspired by the work of graph rewriting and graph grammar researchers.

We regard this work as an on going process in experimenting with our graph rewriting paradigm, and there are many other possible extensions and changes possible. Concerning the modifiers discussed here, experimentation with the modifiers discussed here, extending Once Only to subgraphs is conceivable, given a sensible interpretation and a method of identifying primitives with different labels across LHS graphs.

Acknowledgements

This work was supported by the EPSRC UK research council, grant GR/M23564.

References

1. P.J. Rodgers. A Graph Rewriting Programming Language for Graph Drawing. P.J. Rodgers. *Proceedings of the 14th IEEE Symposium on Visual Languages (VL’98)*. pp. 32-39. 1998.
2. P.J. Rodgers and P.J.H. King. A Graph Rewriting Visual Language for Database Programming. *The Journal of Visual Languages and Computing 8(6)*. Academic Press. pp. 641-674. December 1997.