Context-Exploiting Shapes for Diagram Transformation

Frank Drewes
Institutionen för Datavetenskap, Umeå Universitet, S-90187 Umeå (Sweden)
e-mail: drewes@cs.umu.se

Berthold Hoffmann
Technologiezentrum Informatik, Universität Bremen, D-28334 Bremen (Germany)
e-mail: hof@tzi.de

Mark Minas
IMMD, Universität Erlangen-Nürnberg, D-91058 Erlangen (Germany)
e-mail: minas@informatik.uni-erlangen.de

DIAGEN [1] is a tool for generating diagram editors that respect the *syntax* of particular diagram languages. Editors generated with DIAGEN work as follows:

- A drawing tool allows *free-hand editing* of diagrams, by arranging diagram primitives like boxes, circles, lines, text on a drawing pane. The set of available primitives depends on the particular diagram language.
- A *scanner* analyses the spatial relationships of diagram primitives, and reduces them to *diagram symbols* like states and transitions.
- Finally, a *parser* checks whether the diagram symbols are related according to the syntax of the diagram language, and produces a *derivation structure*.

DIAGEN relies on hypergraphs and hypergraph transformation: diagrams are represented as hypergraphs, the scanner is specified by hypergraph transformation rules, and the syntax of diagram languages is specified by hypergraph grammars. (See [5] for details.) Case studies with editors for control flow diagrams, Nassi-Shneiderman diagrams, message sequence charts, state charts, and UML class diagrams indicate that the syntax of practically every kind of diagram language—also for *design languages* in a more general sense—can be implemented with DIAGEN.

DIAGEN shall now be complemented by DIAPLAN, a diagrammatic, rule-based language for modeling the semantics of diagram languages, e.g. by animating their behavior, or by transforming them into canonical forms. DIAPLAN is based on *hierarchical graphs* wherein the edges may contain graphs in a nested way (see [3]), and on a rather expressive way of graph transformation rules where *graph variables* are used to match, delete, or duplicate subgraphs of arbitrary size [4]. This should make DIAPLAN convenient for developing specialized design tools (but does not restrict it to this purpose). In constrast to standard textual languages, its level of abstraction and its underlying computing paradigm fit the requirements of design tools very nicely.

A central requirement for modern high-level programming languages is the availability of user-defined data types and type checking facilities. In DIAPLAN types are graph languages called *shapes*. This work extends a result of [4] concerning *shaped*

transformations $G \Rightarrow_{L/R} H$ where the hierarchical graphs G, H and the transformation rules L/R adhere to *shape rules* Σ (in symbols: $\Sigma \vdash G$, $\Sigma \vdash H$, and $\Sigma \vdash L/R$):

Theorem 1. *If* $\Sigma \vdash L/R$ *and* $G \Rightarrow_{L/R} H$*, then* $\Sigma \vdash G$ *implies* $\Sigma \vdash H$.

If shape adherance $\vdash$ can be decided, this theorem provides a static shape discipline for transformation sequences $G \Rightarrow_T^* H$ that use shaped rules T: Once the input graph of such a sequence satisfies $\Sigma \vdash G$, we know that its result graph will satisfy $\Sigma \vdash H$. Runtime shape checks (of the intermediate graphs) are not necessary.

In [4], Theorem 1 has been shown for context-free shape rules that replace a single *nonterminal* hyperedge e by a (hierarchical) graph S. (These rules are a straightforward extension of [2] to hierarchical graphs.) Even if this allows to specify rather involved shapes, like that of structured control flow diagrams, it does not suffice to specify the graphs representing all relevant diagram languages.

In the full version of this paper, we will therefore show that theorem 1 can be lifted to *context-exploiting rules*. These rules replace an edge e only if it appears in a certain *context* (subgraph) C, and may exploit C by connecting edges of the replacement graph S to nodes in C. (See Figure 1 for schematic context-free and context-exploiting rules.) Context-exploiting rules allow us to define substantially more involved shapes. As an example, we shall specify the shape of general control flow diagrams (of "spaghetti code") which cannot be specified by context-free rules. As adherence to context-exploiting can be decided by a modification of the parsing algorithm employed in DIAGEN, we still have a static shape discipline.

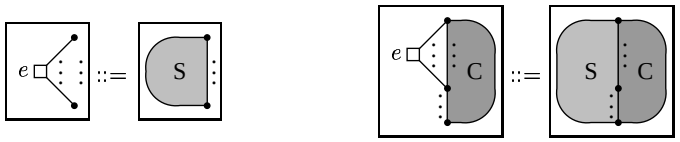

Fig. 1. Schematic context-free shape rules (left) and context-exploiting shape rules (right)

References

1. DIAGEN homepage www2.cs.fau.de/DiaGen/.
2. F. Drewes, A. Habel, and H.-J. Kreowski. Hyperedge replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, chapter 2, pages 95–162. World Scientific, Singapore, 1997.
3. F. Drewes, B. Hoffmann, and D. Plump. Hierarchical graph transformation. *Journal of Computer and System Sciences*, 64:249–283, 2002.
4. B. Hoffmann. Shapely hierarchical graph transformation. In *Proc. IEEE Symposia on Human-Centric Computing Languages and Environments*, pages 30–37. IEEE Computer Press, 2001.
5. M. Minas. Concepts and realization of a diagram editor generator based on hypergraph transformation. *Science of Computer Programming*, 44:157–180, 2002.