Designing Graph Drawings by Layout Graph Grammars

Franz J. Brandenburg

University of Passau
Lehrstuhl für Informatik
94030 Passau, Germany
brandenb@informatik.uni-passau.de

Abstract. Layout graph grammars are a grammatical or rule-based method for the construction of graphs and of their drawings. As such they are representatives of the so-called declarative approach. A layout graph grammar consists of an underlying context-free graph grammar and of layout specifications. These are attached to the productions and consist of position relations and distances between the vertices of the productions. The layout specifications are often derived from drawings of the productions and can be defined in terms of labelled graphs.
Layout graph grammars have been used unnoticed in many examples. They respect the underlying tree-like structure of their graphs and make it visible. This is a new effect, which makes them incompatible with most other graph drawing algorithms. Given a context-free layout graph grammar, there is a polynomial time algorithm which for every specification. The capabilities of layout graph grammars are illustrated by some tree drawings, which range from drawings with quadratic area to area optimal h-v drawings.

1 INTRODUCTION

Graph drawing is concerned with the construction of readable drawings of graphs, i.e. grid layouts satisfying some aesthetics or constraints. The annotated bibliography [8] may serve as a survey. Most contributions to this field are algorithmic and introduce efficient algorithms e.g. for drawings of planar graphs or trees. Recently, hierarchical graphs and descriptions of graphs by expressions have been introduced to cope with the increase in complexity and to describe different levels of abstraction. Here, hierarchical means tree-like or well-formed in terms of expressions and operations. These are not the layered diagrams used for drawings of directed acyclic graphs. Examples of hierarchical graphs are the trees, the series-parallel graphs, flowgraphs of well-structured programs, or graphs obtained by browsing through structures. They all have recursive definitions with simple operations. For example, a binary tree is the empty tree or consists of a node and two subtrees. Series-parallel graphs are built by serial and parallel compositions and flowgraphs along the underlying syntax. Formally and more generally this is captured by graph grammars.

We introduce a formal framework for the hierarchical design of graphs and of their drawings. It is based on graph grammars, which are canonical extensions of context-free string grammars. There is a rich theory of graph grammars and of graph languages, see e.g. [4, 11, 12, 18]. We follow the algorithmic or vertex replacement approach, which is best suited for our purpose. In general, a graph grammar generates infinitely many graphs which are comprised into the graph language. For effective manipulations on graph grammars and feasible i.e. polynomial time complexity the restriction to so-called context-free graph languages seems necessary. This has strong implications, since the grids and the planar graphs cannot be captured by context-free graph grammars. This is a serious drawback of the approach.

A graph grammar consists of a finite set of productions of the form (A, R, C), where A is a vertex label, R is a finite graph and C is the connection relation. In a derivation step, a vertex w with label A is replaced by the graph R and C establishes edges between the neighbors of w and the vertices of R. For a layout graph grammar the designer creates finitely many drawings of every production and specifies which of the shown position relations and distances are relevant. This should be supported by an editor, as in the HiGraD [1] and Graph^{Ed} systems [15].

The design of a layout graph grammar and also of graph grammars is much easier as it seems at the first glance. There is a natural graphic representation of the productions (A, R, C). See the figures below. Draw the right-hand side graph R such that it looks nice and expresses the intended meaning. For every production there may be several such drawings. The vertices are drawn as squares of (almost) unit size or are points in the plane. The edges of R are drawn as straight lines. The left hand side A is drawn as a big rectangle which includes the drawing of R. The tuples from the connection relation C are represented by edges from a vertex label outside the rectangle to one of its sides and from there to the specified vertices of the right hand side R.

In a top-down view, this describes the refinement of a vertex with label A by the graph R from the right hand side. In a bottom-up view, the subgraph derived from R is enclosed by a big rectangle for the left hand side. This rectangle reserves the space needed for the derived subgraph. Its final size and the positions of the vertices placed inside are computed by attributation techniques, which need only linear time, if the derivation is given. If the underlying graph grammar can be parsed in polynomial time by some common algorithm, then we can compute the area optimal drawings of the generated graphs in polynomial time. For some well-chosen examples these drawings coincide with those obtained by special algorithms.

Layout graph grammars were already used before, but they were not recognized as such. Examples are the drawings of complete binary trees as H-trees [20], drawings of series-parallel graphs, diagrams of finite state automata, [20] and [16, Fig. 2.13 - Fig. 2.15] well-structured flowgraphs and h-v drawings of binary trees [7, 10]. This shows that our approach is natural. Some specialized types of layout graph grammars have been implemented in the Graph^{Ed} [15] and HiGraD [1] systems.

Related approaches have been introduced in [13, 17, 21]. Our goal is to optimize over all possible derivations, and thus to use layout graph grammars for syntax directed constructions of graphs and their drawings.

2 BASIC NOTIONS

We consider directed or undirected graphs with labelled vertices and labelled edges. Multiple edges with different labels are allowed and are needed. The labels are not important for the final graphs. However, they are necessary for graph grammars and are used for a graph theoretic specification of the graph layouts. Thus, labelled graphs provide a uniform graph theoretic framework, both for the graphs and their layouts.

Definition 1. Let Σ and Δ be finite sets of vertex labels and of edge labels. A directed labelled *graph* $g = (V, E, m)$ over Σ and Δ consists of a finite set of vertices V, a vertex labelling function $m : V \to \Sigma$ and a finite set of directed and labelled edges $\mathrm{E} = \{(u, a, w) \mid u, w \in V, u \neq w \text{ and } a \in \Delta\}$. An undirected edge is identified with a pair of directed edges in both directions. An edge (u, a, w) is called an a-edge from u to w. When necessary, we write $g = (V(g), E(g), m(g))$.

Let $\Delta' \subseteq \Delta$ be a set of distinguished edge labels. This selection induces a canonical partition of a graph $g = (V, E, m)$ into graphs $g_{\Delta'}$ and $g_{\Delta - \Delta'}$ with the same sets of vertices, such that $g = g_{\Delta'} \cup g_{\Delta - \Delta'}$. Here $g_{\Delta'} = (V, E \cap \{(u, a, w) \mid a \in \Delta')\}, m)$ is the restriction of g to Δ'. Conversely, define the union of graphs with the same sets of vertices and disjoint sets of edge labels by taking the union of the sets of edges. Edge labels are also used for special paths. A path consisting of directed a-edges only is called an a-path.

Vertices u and w are neighbors, if they are connected by an edge. The neighbors of a vertex w can be distinguished by their vertex labels and by the direction and the label of their connecting edges. This is important for graph grammars. For a vertex label B, an edge label a and the direction "in" (resp. "out") the (B, a, in)-neighbors of a vertex w are the vertices $\{u \in V(g) \mid m(u) = B$ and there is an edge (u, a, w) in $E(g)\}$. Such triples (B, a, in) and similarly (B, a, out) define classes of neighbors of a vertex w, which are indistinguishable for graph grammars.

Definition 2. A *drawing* of a graph $g = (V, E, m)$ or its *layout* is a mapping $d(g)$ into the discrete plane or grid. d maps each vertex $w \in V$ to a square of (almost) unit size with the center at a grid point such that the images of distinct vertices do not overlap. Such squares may degenerate to points. Each edge $e = (u, a, w)$ from u to w is mapped into a curve from $d(u)$ to $d(w)$. We consider only straight line drawings. These can be extended to polyline drawings, if the inner points $p_2, \ldots, p_{n-1}$ of a polyline $d(e) = p_1, \ldots, p_n$ are treated as extra vertices with special vertex labels.

Our approach to graph drawings is based on graph grammars. The core of a graph grammar is its finite set of productions. These are constructed by a user or a designer and so are fully under his control. The productions are used both for the generation of the graphs and for their drawings. This requires a description of the transfer of the layout information from the static set of productions to the dynamic derivation processes and the generated graphs.

Definition 3. A drawing specification $DS(g)$ of a graph $g = (V, E, m)$ consists of a set of position relations and distances between the vertices of g. For a vertex w let $w.X$ and $w.Y$ denote the values of the X- and Y-coordinates associated with w under some drawing d.

For a pair of vertices (u, w) define a "left-to-right" relation and minimal distances in X-dimension by inequalities of the form $w.X > u.X + k$ where $k \geq 0$ is a natural number. The interpretation of $w.X > u.X + k$ is that in every drawing, w is placed at least $k+1$ units to the right of u. Similarly, define "upwards" relations and minimal distances in Y-dimension by $w.Y > u.Y + m$. Clearly, such relations between vertices do not fix their distance and the direction from u to w, even if both are given. If u is at $(-k-1, -m-1)$ then w can be anywhere in the first quadrant. Conversely, every drawing makes the directions and the distances tight and converts the inequalities into equations.

Moreover we introduce horizontal and vertical alignments. This extends our earlier approach [3]. Alignments are defined by equations of the form $w.Y = u.Y$ resp. $w.X = u.X$ with the obvious meaning that u and w have the same Y-coordinates resp. X-coordinates. Clearly, for $u \neq w$, both $u.Y = w.Y$ and $u.X = w.X$ are forbidden, since this means an overlap of the vertices u and w.

This definition is too loose. A drawing specification of a graph g should provide sufficient information for the construction of nice drawings. A basic requirement is that the drawing specification guarantees no overlap of the vertices. This is obtained by the restriction that for every pair of vertices a left-to-right relation or an upwards relation must be specified. A drawing specification does not describe a routing of the edges which are drawn as straight-lines. These may pass through other vertices or may cross each other. Can this be excluded? Are there other meaningful drawing specification in the context of layout graph grammars? This is still open.

There is an alternative and purely graph theoretic definition of a drawing specification $DS(g)$ of a graph $g = (V, E, m)$. Let x, y, h, v be new edge labels. For every pair of vertices (u, w) a left-to-right relation with $w.X > u.X + k$ is represented by a directed x-edge (u, x, w) from u to w with length k. Accordingly, represent upwards relations with $w.Y > u.Y + m$ by a directed y-edge (u, y, w) with length m. The length is attached to the edge and is omitted if the length is one. Horizontal resp. vertical alignments are expressed by undirected h-edges resp. v-edges.

The set of all left-to-right relations is comprised into a graph $g_x = (V, E_x, m)$, where V is the set of vertices of g and E_x are the x-edges. g_y, g_h and g_v are defined accordingly. Let $g^* = g_x \cup g_y \cup g_h \cup g_v$. Then $f = g \cup g^*$ is a graph consisting of the underlying graph g together with its layout specification g^*.

We wish to simplify the approach and focus on the essentials. Lets assume that the minimal left-to-right and upward distances between vertices are one unit. Then these relations are expressed by greater than relations of the form $w.X > u.X$ resp. $w.Y > u.Y$ and by x-edges resp. y-edges in the graph theoretic definition. The general case can be obtained by attributations and simple calculations.

Vertices shall not overlap. Geometrically the placement of two vertices must differ in the X- or in the Y-coordinate. In the graph theoretic setting this leads to the following definition.

Definition 4. Let g be a graph and let $g^* = g_x \cup g_y \cup g_h \cup g_v$ be a drawing specification in terms of z-edges with $z \in \{x, y, h, v\}$. The drawing specification g^* is consistent if

1. g_x and g_y are acyclic.
2. for every pair of vertices (u, w) with $u \neq w$ there is an x-path from u to w, or conversely, or a y-path from u to w, or conversely.
3. The undirected graph g_h consist of cliques, and similarly for g_v.
4. For every pair of vertices, h-edges and y-paths, v-edges and x-paths and h-edges and v-edges between them exclude each other.

These restrictions can be transferred to graph grammars.

3 GRAPH GRAMMARS

Next we give a short introduction to graph grammars and illustrate some highlights. See for example [4, 5, 6, 11, 12, 18] for basic notions and results. We follow the set theoretic or algorithmic approach and consider graph grammars from the family of vertex replacement systems. This approach is best suited for graph drawing, although it has strong limits, as we will see below.

Definition 5. A *graph grammar* is a system $GG = (N, T \cup \Delta, P, S)$, where N and T are the alphabets of nonterminal and terminal vertex labels and Δ is the alphabet of terminal edge labels. S is the axiom and is regarded as an S-labelled vertex. P consists of finitely many productions of the form $p = (A, R, C)$. Here $A \in N$ is the label of a nonterminal vertex w, R is a nonempty, labelled graph of the right-hand-side and C is the connection relation. For simplicity, let C preserve the direction and the edges labels. C distinguishes incoming and outgoing edges and consists of tuples (B, a, u) with $B \in (N \cup T), a \in \Delta$ and $u \in V(R)$. A tuple (B, r, u) establishes edges between the specified neighbors of w and the new vertex u from R.

There is an elegant graphic representation of the productions, which is taken as the starting point for our layout graph grammars. Consider fig. 1. The graph grammar has two productions. It generates the binary trees over the terminal vertex label a. The right-hand side graph of the first production (A, R, C) is a

tree with three vertices whose root is labelled by a and the two leaves are labelled by A. The directed edges go from the root to the leaves. They are unlabelled. The left hand A is drawn as a rectangle, which tightly encloses the tree from the right-hand side. There is a single tuple (a, a) in the connection relation, which is shown by the edge from the vertex label a on top of the rectangle to the root. The second production replaces a vertex with label A by a vertex with the terminal label a preserving the connections from a. This is a renaming of A into a. See [3] for further examples. This representation is self-explaining and it helps in understanding graph grammars and their derivations.

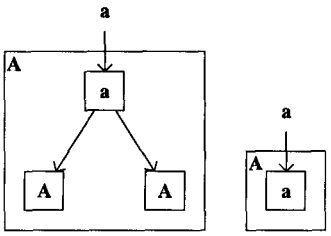

Fig.1

If a production (A, R, C) is given in textual form it can be converted into a visual form as follows. First draw the right-hand side graph R in some way, such that its vertices are squares of unit size and its edges are straight lines. Place the vertices in such a way that they do not overlap. Simply speaking, let $d(R)$ be some drawing of R. Then draw the left-hand side A as a big rectangle, $box(p)$, such that $box(p)$ includes the drawing of R. $box(p)$ has an expansion $width(box(p))$ in X-dimension and $height(box(p))$ in Y-dimension. Finally, the connection relations (B, a, u) are represented by lines, which start from a virtual point with label B outside $box(p)$, they cross the rectangle $box(p)$ at a distinguished entry point and are then routed inside $box(p)$ to the designated vertex u of R. This procedure fully describes the task of a designer of a layout graph grammar.

Definition 6. A direct *derivation step* $g \Longrightarrow g'$ rewrites a graph g into some graph g' by the application of some production p at some vertex w. This means an incremental change. To obtain g' from g using the production $p = (A, R, C)$ select a vertex w of g with label A. Replace w by an isomorphic copy of R that is vertex disjoint with g. Then establish edges between the neighbors of w and the vertices of R as specified by C. Formally, $g' = (V', E', m')$, where $V' = V(g) - \{w\} \cup V(R)$. The vertex labels are copied from g and R. An edge (s, a, t) is in E' if and only if $e \in E(g)$ and $s \neq w \neq t$ or $e \in E(R)$ or e is established by a connection from C as follows. Consider incoming edges; outgoing edges are similar. If $(B, a, u) \in C$ and $u \in V(R)$, then (v, a, u) is an edge of g' if and only if (v, a, w) is an edge in g. The *language* generated by a graph grammar consists of all generated graphs with terminal vertex labels, $L(GG) = \{\ g \mid S \Longrightarrow^* g$ with $m(w) \in T$ for every vertex $w \in V(g)\}$.

Our view to graph grammars and to layout graph grammars is dominated by the "inclusion principle", which is expressed by the fact that the graph of the

right-hand side R is included in the left-hand side A. This concept corresponds to the "contains operation" of Golin and Reiss [13].

There is a rich theory of graph grammars and graph languages that has been developed in the past decade, see [4, 5, 6, 11, 12, 18]. Many investigations are concerned with the proper notions of context-free graph grammars and context-free graph languages. There exist several concepts, such as confluent and boundary graph grammars and hyperedge replacement systems, but that is not of concern, here. We shall only rely on effective parsing algorithms, such as those in [2, 18], and these should run in polynomial time.

We cannot go into depth and details and state some important facts and highlights:

Proposition 7. *The following sets of graphs can be generated by context-free graph grammars and thus are context-free graph languages. See e.g. [18, 11].*

- *the trees and the binary trees*
- *the series parallel graphs*
- *the partial k-trees for fixed k*
- *the maximal outerplanar graphs*
- *the (complete) bipartite graphs*
- *the complete graphs*
- *the flowgraphs of programs.*

These sets of graphs have a recursive definition using some operations, and each graph has an underlying tree structure. To the contrary, there are many sets of graphs which cannot be defined by context-free graph grammars. There is a *Pumping Lemma* which implies a linear growth in the size of the generated graphs and a *separator theorem* [19] which says that every generated graph as a separator which is quadratic in the degree of the graph. This excludes e.g. the square grids and many other sets of graphs which are of special interest for graph drawings.

Proposition 8. *The following sets of graphs cannot be generated by any context-free graph grammar and thus are non-languages. Even more, these sets of graphs are not contained in any context-free graph language [19].*

- *the grids*
- *the planar graphs*
- *the directed acyclic graphs*

4 LAYOUT GRAPH GRAMMARS

Layout graph grammars transfer the drawing specifications of finitely many productions into infinite derivation processes. They produce graphs together with their drawing specifications. This can fully be described in graph grammar terms. Hence, layout graph grammars perform a syntax directed translation of textual

representations of graphs into graph drawings. Alternatively they can be seen as attributed graph grammars, where the attributes describe geometric relations. Attributation techniques for graph drawings are also used by Golin and Reiss [13] and by Zinßmeister [21].

Definition 9. A *layout graph grammar LGG* consists of a graph grammar GG and a layout specification LS. LS associates finitely many drawing specifications $c_1, \ldots, c_q$ with each production p of GG with $q \geq 1$.

A pair (p, c) is called a *layout production*. (p, c) is applied to a pair $(g, DS(g))$ consisting of a graph g and a drawing specification of g. It yields a new pair $(g', DS'(g'))$ by an incremental change. g' is obtained from g by replacing a vertex w of g by the graph R according to p. The drawing specification is updated as follows. Consider only the horizontal updates; the vertical ones are similar. Let $k = width(box(p))$ be a lower bound of the size of $box(p)$ in X-dimension derived from the drawing specification c. Then $DS'(g')$ includes all inequalities from the layout specification of the right hand side R. The inequalities of $DS(g)$ with the replaced vertex w are updated and are transferred to all vertices of the right hand side. Thus the left side of the rectangle $box(p)$ inherits all "to the left of" relations of w. Hence, if $w.X > z.X + k$ holds in $DS(g)$ for some vertex $z \in V(g)$ and $u.X > h.X + m$ holds in c where h represents the left side of $box(p)$ and u is some vertex of R, then $u.X > z.X + (k + m)$. Hence, u must be drawn more than $k + m$ units to the right of z, if w was at least k units to the left of z and u is at least m units to the right of the left border of the enclosing rectangle. Similarly, the right side of $box(p)$ inherits all "to the right of" relations from w. The expansion of the replaced vertex w to $box(p)$ may have an impact on the relations between other vertices of g, which are updated by adding an extra shift. Layout specifications for horizontal or vertical alignments between the replaced vertex w and another vertex z are transferred to distinguished vertices of the right hand side. If $w.X = z.X$ is specified in $DS(g)$ then $w.X = u.X$ holds for every distinguished vertex.

With the simplification to unit distances the layout specifications can be expressed in graph grammar terms using the extra edge labels $\{x, y, h, v\}$.

Definition 10. Let $p = (A, R, C)$ be a production and let R_x, R_y, R_h and R_v be the graphs with vertices $V(R)$ describing a drawing specification c. Then $(A, R \cup R^*, C \cup C^*)$ describes a layout production (p, c) where $R^* = R_x \cup R_y \cup R_h \cup R_v$ and C^* handles the connections of x, y, h, v - edges . In a derivation step, incoming x-edges of the replaced vertex w are transferred to all x-sources of g_x and outgoing x-edges to all x-sinks. y-edges are treated accordingly. For the h-edges resp. v-edges, there are distinguished vertices in R and these inherit the h-edges resp. w-edges from the replaced vertex w.

Definition 11. The *language* $L(LGG)$ of a layout graph grammar $LGG = (GG, LS)$ consists of pairs $(g, DS(g))$ with $g \in L(GG)$ and $DS(g)$ is constructed along a derivation $S \Longrightarrow^* g$. Equivalently, $L(GG)$ consists of graphs of the form $f = g \cup g^*$. Let $D(LGG) = \{d(g) \mid (g, DS(g)) \in L(LGG)$ and the drawing

$d(g)$ satisfies the specification $DS(g)\}$. $D(LGG)$ contains the drawings of the generated graphs, which satisfy the layout specification.

Every derivation step by a production p induces an expansion by at most $width(box(p))$ and $height(box(p))$. Since context-free graph grammars permit derivations of graphs of linear length, this implies a linear upper bound in each dimension. Recall that there is only a placement of the vertices.

Lemma 12. *Every drawing $d(g)$ of $D(LGG)$ has an area of size $0(n^2)$, where n is the size of g.*

The usefulness of the approach is based on the following result.

Theorem 13. *Let $LGG = (GG, LS)$ be a layout graph grammar such that GG is polynomial, i.e. there is a polynomial algorithm for the membership problem $g \in L(GG)$ as in [2, 17]. Then there is a polynomial time algorithm H which for every graph $g \in L(GG)$ constructs a drawing $d(g)$ such that $d(g)$ satisfies a specification $DS(g)$ with $(g, DS(g))$ in $L(GG)$ and $d(g)$ has minimal area among all these drawings.*

Proof. (Sketch). By definition, the underlying graph grammar has a polynomial time membership problem and every graph $g \in L(GG)$ can be parsed in polynomial time. Consider a derivation $\delta : S \Longrightarrow^* g$ which is reconstructed by the parsing algorithm. For every vertex w occurring in d define attributes $width(w)$, $height(w)$, $X(w)$ and $Y(w)$. width and height are synthesized attributes. They define the minimal size of a rectangle containing the graph generated from w. X and Y are inherited attributes defining the X- and Y-coordinates of w relative to the next enclosing rectangle. These values are computed in a final top-down pass. If a vertex w is replaced by a graph R using a production (A, R, C), then for each layout production (p, c) we can compute the updates of $width(v)$ and $height(v)$ from the associated values for the vertices of R and from the drawing specification c. The parsing algorithm for the underlying graph grammar is augmented with these attributes. It computes minimal values for the width and the height of each vertex. Since width and height are bounded by $O(n)$, there are at most $O(n^2)$ incomparable pairs, where n is the size of g. This implies a quadratic increase in the running time of the parsing algorithm, which is supposed to be polynomial.

The outcome of algorithm H depends on the used layout specifications and thus on the layout graph grammar designed by the user. There is a wide spectrum which is illustrated here for binary trees. Let the vertices be u, w, v from left to right.

The production in fig. 2a is used to define binary trees and to construct H-tree layouts, see [20]. For completeness add a production which relabels vertices from A to a, as in fig. 1. The layout specification can directly be retrieved from the drawing. In the graph theoretic definition of the drawing specification, the tree edges coincide with h-edges resp. v-edges as shown. The transitive reductions of

the x-edges and the y-edges is shown in figures 2b and 2c. The y-edges to the center vertex could be omitted; they are implied by the vertical y-edges and the h-edges. With this layout graph grammar, algorithm H constructs area optimal H-tree layouts.

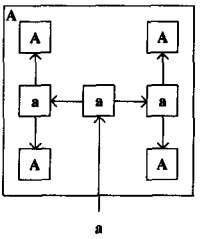

Fig.2a

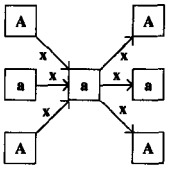

Fig.2b

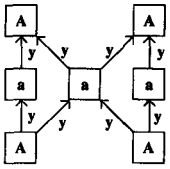

Fig.2c

Next, consider drawings of binary trees from top to bottom using the graph grammar from fig. 1. Consider only the left production, the right one is trivial. If the left-to-right and upwards relations between the three vertices of the right-hand side are as shown then algorithm H produces inorder drawings. If t is a generated tree with n vertices and height h then the X-coordinates of its vertices are given by the inorder numbering. The height of the drawing coincides with the height of the tree, because the root is at least one unit above its children. However, the Y-coordinate of a lower subtree is not exactly fixed. It may vary so that t fits into the rectangle of size $n * h$, which is the area used by algorithm H.

Now add a horizontal alignment between the leaves of the production, i.e. an h-edge between the two A-vertices. If in derivation steps this alignment or h-edge is inherited to the root, then algorithm H produces proper inorder drawings of binary trees, where for every tree node t the X-coordinate is determined by the inorder numbering and the Y-coordinate by the distance from the root. However, if the horizontal alignment or h-edge is inherited to the (left) leaf of the production then all leaves are positioned on the same horizontal level. They lie on a line. The Y-coordinates of the interior nodes are not exactly fixed. The area is $n * h$.

If alternatively a y-edge is added from the left leaf to the right leaf, then the left subtree must be drawn completely below the right subtree. This results in tree drawings of (maximal) width and height n for trees of size n.

The aforementioned tree drawings are simple and even bad. They cannot compete with those of tree drawing algorithms, because the drawings are too wide. This is improved by our last example which uses two drawing specifications for a single production.

Take the specification as shown in Fig. 3 where the edge labels indicate the specification. Notice that the two drawing specifications differ only by the (diagonal) edge between the two leaves, which alternatively is an x-edge or a y-edge. In derivation steps the horizontal and vertical alignments are inherited to the root. Using both layout productions, algorithm H is the graph grammar version of the algorithm from [10]. It constructs area optimal h-v drawings of

binary trees and runs in quadratic time. For complete binary trees and Fibonacci trees it constructs h-v drawings with linear area as in [8], however it needs quadratic time.

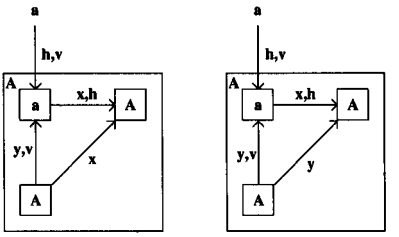

Fig. 3

These notes illustrate that tree drawings by graph grammars cannot compete with tree drawing algorithms. E.g. our graph grammars do not specify that the root is centered over the subtrees; such features can be added. There is a proper left-ro-right or upwards relation between subtrees. These cannot be merged. Hence subtrees cannot be squeezed along their contour, as in many tree drawing algorithms.

5 CONCLUSION

Layout graph grammar are a complex and powerful concept. However, are they really useful for graph drawings? They cannot deal with grids or planar graphs! They focus only on the placement of the vertices; the routing of the edges is reduced to straight lines. However, there is no guarantee that such lines cross vertices others than their endpoints. Extensions and modifications should overcome these limitations.

6 ACKNOWLEDGEMENT

I wish to thank the referees for their critisism and useful comments.

References

[1] W. Bachl, F.J. Brandenburg, T. Hickl *"Hierarchical Graph Design Using HiGraD"* Technical Report, University of Passau, MIP 9405 (1994)

[2] F.J. Brandenburg *"On polynomial time graph grammars"* Lecture Notes in Computer Science 294 (1988), pp. 227-236

[3] F.J. Brandenburg *"Layout Graph Grammars: the Placement Approach"* Lecture Notes in Computer Science 532 (1991), pp. 144-156

[4] B. Courcelle *"Graph rewriting: An algebraic and logic approach"* Handbook of Theoret. Comput. Science, Vol. B, Elsevier (1990), pp. 193-242

[5] B. Courcelle *"On the structure of context-free sets of graphs generated by vertex replacement "* Research Report 91-44, Bordeaux University (1991)

[6] B. Courcelle, J. Engelfriet *"A logical characterization of the sets of hypergraphs defines by hyperedge replacement grammars"* Research Report 91-41, Bordeaux University (1991)

[7] P. Crescenzi, G. Di Battista, A. Piperno *"A note on optimal area algorithms for upward drawings of binary trees"* Comput. Geometry: Theory and Applications 2 (1992), pp. 187-200

[8] G. Di Battista, P. Eades, R. Tamassia, I. Tollis *"Algorithms for Drawing Graphs: an Annotated Bibliography"* Technical Report, Brown University, Providence (1993)

[9] P. Eades, T. Lin *"Algorithmic and declarative approaches to aesthetic layout"* Proc. Graph Drawing '93 (1993), pp. 95-96

[10] P. Eades, T. Lin, X. Lin *"Minimum Size h-v Drawings"* Proc. Advanced Visual Interfaces (1992) World Scientific Series in Computer Science Vol. 36, pp. 386-394

[11] H. Ehrig, H.-J. Kreowski, G. Rozenberg (Eds.) *"Graph Grammars and their Application to Computer Science"* Lecture Notes in Computer Science 532 (1991)

[12] J. Engelfriet *"Context-free NCE graph grammars"* Lecture Notes in Computer Science 380 (1989), pp. 148-161

[13] E.J. Golin, S.P. Reiss *"The Specification of Visual Language Syntax"* J. Visual Languages 1 (1990), pp. 141-157

[14] T. Hickl *"Rechtwinkeliges Layout von hierarchisch strukturierten Graphen"* University of Passau (1994)

[15] M. Himsolt *"Konzeption und Implementierung von Grapheneditoren"* Dissertation, University of Passau (1993)

[16] J.E. Hopcroft, J.D. Ullman *"Introduction to Automata Theory, Languages and Computation"* Addison/Wesley (1979)

[17] C.L. McCreary, C.L. Combs, D.H. Gill, J.V. Warren *"An automated graph drawing system using graph decomposition"* Proc. Graph Drawing '93 (1993), pp. 119-120

[18] G. Rozenberg, E. Welzl *"Boundary NLC graph grammars - basic definitions, normal forms and complexity"* Inform. Control 69 (1986), pp. 136-167

[19] R. Schuster *"Graph Grammatiken und Graph Einbettungen: Algorithmen und Komplexität"* Technical Report, University of Passau, MIP 8711 (1987)

[20] J. D. Ullman *"Computational Aspects of VLSI"* Computer Science Press (1984)

[21] G. Zinßmeister *"Layout of trees with attribute graph grammars"* Proc. Graph Drawing '93 (1993), pp. 99-102