Annales UMCS Informatica AI VIII, 2 (2008) 89–96
DOI: 10.2478/v10065-008-0027-z

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Hierarchical graph transformations with meta-rules

Wojciech Palacz*

Department of Computer Design and Graphics, Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University,
ul. Reymonta 4, 30-059 Kraków, Poland

Abstract

This paper is concerned with hierarchical graph models and graph transformation rules, specifically with the problem of transforming a part of graph which may contain subordinated nodes and edges. Meta-rules are proposed as a formal way of representing transformations which remove or duplicate a node along with its contents. The paper discusses the behaviour of meta-rules when applied to different types of hierarchical graphs, possible failure cases, and concludes by introducing a type of hierarchical graphs in which meta-rules can always be successfully expanded.

1. Introduction

Our research group is interested in computer support systems for the preliminary phase of the design process. In this phase, the designer works on a fairly high level of abstraction, considering the decomposition of the whole object into subcomponents. Hand–drawn sketches similar to the one displayed in Fig. 1 are traditionally used to capture design decisions.

* *e-mail address:* wojciech.palacz@uj.edu.pl

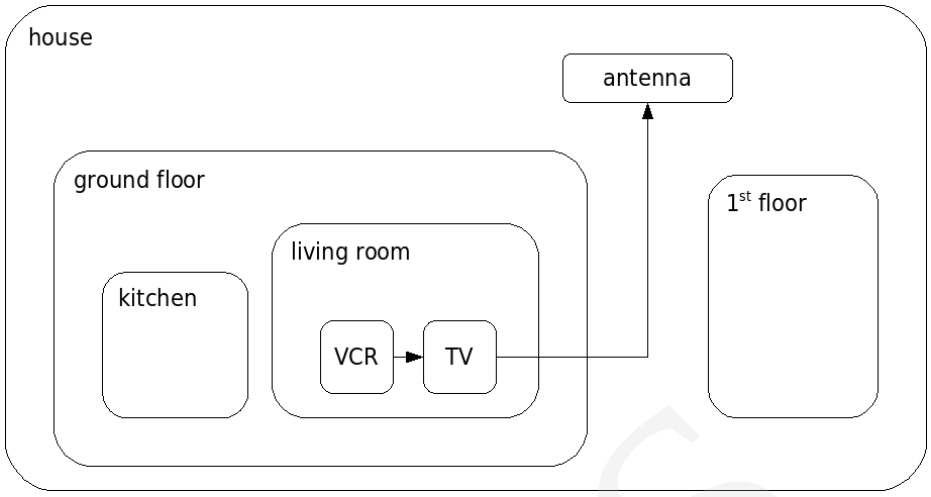

Fig. 1. Structural model of a house

In order to store such sketches in computer memory, an appropriate data model must be utilized. Graphs are the most obvious choice. Our group has defined over the years several formalized graph models meant for this purpose; experiences gained suggest that the model used should be hierarchical, so that it can directly represent hierarchical (sometimes even recursive) nature of real-world objects. Additionally, used formalism should provide some kind of graph transformation rules, which allow for automatic generation and modification of designs.

A generic framework for hierarchical graphs was developed and presented in [**1, 2**]. It provides a way for constructing graphs where graph atoms (nodes, edges, and hyperedges) can be nested inside any other atom, and extends the well–known double–pushout graph transformation rules so that they can be applied to the hierarchical graphs.

Our example uses directed edges and labeled nodes; thus, the formal definition of the hierarchical graphs used by the floor layout design support system should be as follows:

Definition 1. Graph G is a tuple (V, E, s, t, lab, par), where:
• V and E are the finite and disjoint sets of nodes and edges;
• $s\colon E \to V$ and $t\colon E \to V$ are the edge attachment functions (source and target);
• $lab\colon V \to L$ is a node labeling function (L is a set of labels);
• $par\colon V \cup E \cup \{\perp\} \to V \cup E \cup \{\perp\}$ is a parent assigning function;
• $par(\perp) = \perp$, $\forall a \in V \cup E \, \forall k \in \mathbb{N}_+ : par^k(a) \neq a$.

The special symbol $\perp$ is used to denote that a given node or edge has no parent (i.e. is at the top level of the graph). Conditions specified in the definition ensure acyclicity of the parent function, which means that the parent-child relations between graph atoms are a tree with $\perp$ as its root.

In addition to graphs we also need to define graph morphisms. They are mappings between two graphs, mapping nodes to nodes and edges to edges while preserving all their properties (edge attachments, labels, etc.). For graphs specified by Definition 1, a corresponding morphism definition looks like this:

Definition 2. Morphism between two graphs G and H is a function $f\colon V_G \cup E_G \cup \{\perp\} \to V_H \cup E_H \cup \{\perp\}$, where:
- $\forall v \in V_G\colon f(v) \in V_H$, $\forall e \in E_G\colon f(e) \in E_H$;
- $\forall v \in V_G\colon lab_G(v) = lab_H(f(v))$;
- $\forall e \in E_G\colon f(s_G(e)) = s_H(f(e))$ and $f(t_G(e)) = t_H(f(e))$;
- $\forall a \in V_G \cup E_G\colon f(par_G(a)) = par_H(f(a))$.

Morphisms are usually denoted simply as $f\colon G \to H$. If f is both surjective and iniective, then f is an isomorphism; in many practical applications two isomorphic graphs are considered to be the same graph.

It can be demonstrated that graphs from Definition 1 and morphisms from Definition 2 constitute an algebraic category (see [**1**] or [**2**]). Thus, it is possible to define double–pushout rules. This paper will introduce them in an informal way; see [**3**] for the classical definition and exhaustive discussion of their properties, and [**1, 2**] for the definition of hierarchical rules and corresponding proofs.

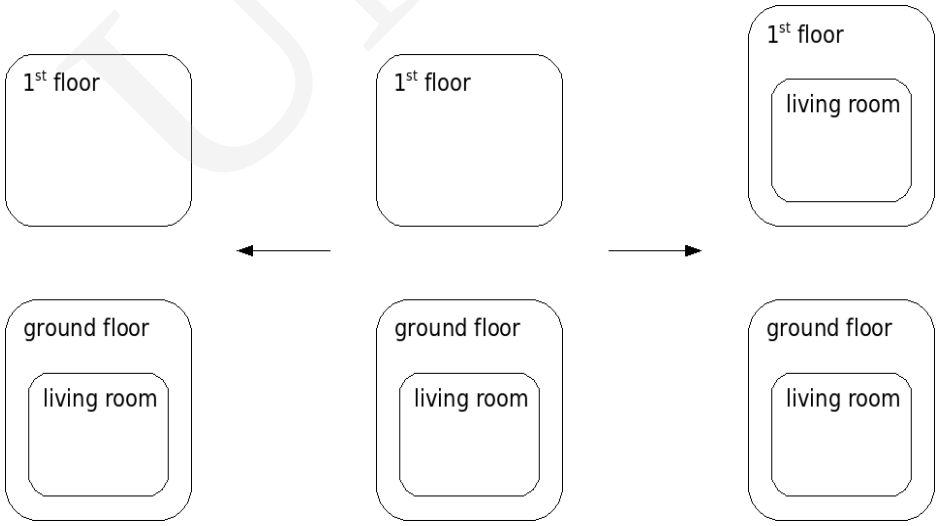

Fig. 2. A double–pushout rule which duplicates a node

A double–pushout rule consists of three graphs (known as the left–hand side, the interface, and the right–hand side, usually denoted by L, K, R) and two morphisms. Morphisms (denoted as $l\colon K \to L$, $r\colon K \to R$) map the interface to the left– and right–hand side graphs, respectively. It can be said that the interface is the intersection of both sides.

Fig. 2 displays an example rule. In order to apply it to the graph presented in Fig. 1 (let us denote it by G), an occurrence of the left–hand side must be found in graph G, i.e. a morphism $m\colon L \to G$ must be found (there is exactly one match possible). Next, atoms present in L but not in K are deleted from G (in our example there are none), and then atoms present in R but not in K are added to G (there is one such node). The obtained result graph will contain a new living room on the 1$^{\text{st}}$ floor.

2. Meta–rules

Simple rules can transform only these graph atoms which were matched to the left–hand side. Operations like removal or duplication of a given node along with its unknown subordinated nodes and edges are beyond their power. This is unfortunate, because such operations are often necessary – for example, a typical user of a design support system for floor layouts will expect to be able to copy, move and delete rooms along with their contents. Meta–rules are the proposed solution. They contain markers, which are matched to subordinated nodes and edges; by deleting or duplicating markers these atoms (unknown in advance) are deleted or copied.

Definition 3. Graph with markers is a tuple $(V, E, s, t, lab, par, mark)$, where:
• (V, E, s, t, lab, par) is a graph (see Definition 1);
• $mark\colon V \cup E \to PR(M)$ is a marker assigning function (M is a set of markers, $PR(M)$ is the set of subsets with repetitions of M).

For a given atom a, $mark(a)$ may be empty. Every graph with markers can be trivially converted to the corresponding simple graph by discarding the marker function.

Definition 4. Graph transformation meta–rule consists of three graphs with markers (L, K, R) and two morphisms (l, r) such that:
• after discarding markers $L \leftarrow K \rightarrow R$ is a transformation rule;
• $mark_L(a)$ is empty or contains exactly one marker for all $a \in V_L \cup E_L$;
• all markers used in L are different;

- $mark_K(a) \subset mark_L(l(a))$ for all $a \in V_K \cup E_K$;
- markers used in R must be also present in L.

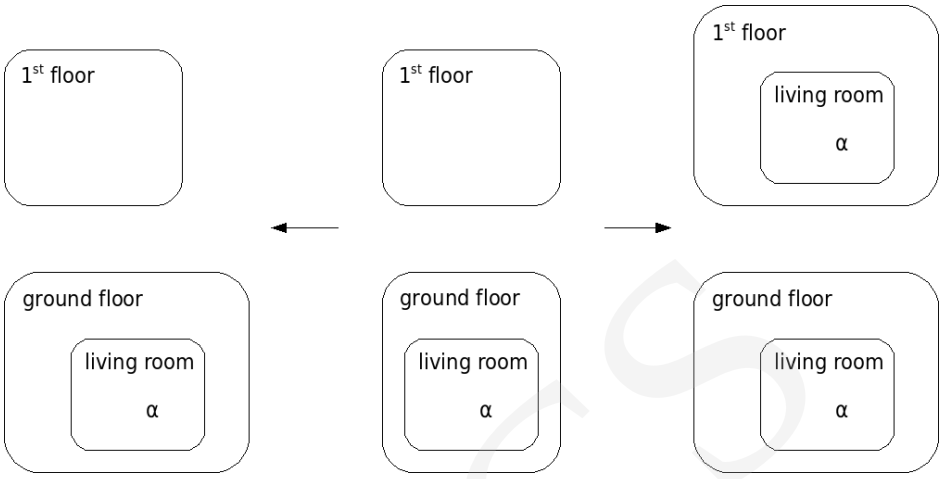

Fig. 3. A meta-rule for duplicating rooms with their contents

Meta–rules are not applied directly. Instead, their markers are expanded to obtain derived rule specific to the graph being transformed, and then this derived rule is applied.

Definition 5. If $m\colon L \to G$ is a morphism matching a meta–rule to a graph being transformed, and α is a marker nested in an atom a from L, then expansion set of this marker is defined as $\exp(\alpha) = ch^*_G(ch_G(m(a)) - m(ch_L(a)))$. Function $ch\colon V_G \cup E_G \cup \{\perp\} \to P(V_G \cup E_G)$ is the children function, i.e. the inversion of the parent function.

Please note that $\exp(\alpha)$ is called "expansion set", not "expansion subgraph". A set of nodes and edges from graph G is a subgraph of G if and only if the following conditions are met:

- if edge e is in $\exp(\alpha)$, then $s(e)$ and $t(e)$ are in $\exp(\alpha)$;
- if atom a and its ancestor $par^k(a)$ are in $\exp(\alpha)$, then $par(a)$ is in $\exp(\alpha)$.

A rule can be derived from a meta–rule if and only if all marker expansion sets are subgraphs. The derivation algorithm works in a loop, eliminating markers one by one. While there are markers in L, the following steps are executed:

1. Choose a marker from L – let us assume it is α, nested in atom a.
2. Delete α from L and insert a copy of $\exp(\alpha)$ in its place, extend morphism m with identity function so that atoms of this copy are mapped onto the original atoms in G.

3. Check if α is present in K. If yes, then it has to be in $l^{-1}(a)$. Replace it with $\exp(\alpha)$, extend morphism l with identity mapping so that newly inserted copy of $\exp(\alpha)$ in K is mapped onto the copy in L.
4. If the condition in the previous step was true, then check R to see if α is present in $r(l^{-1}(a))$. If yes, replace it and extend morphism r as in the previous step.
5. Replace all occurrences of α in R with $\exp(\alpha)$, do not modify r.

The correctness of this algorithm can be proved by observing that the results produced by every step are valid meta–rules; that because of conditions in Definition 4 eliminating all markers from L means that all markers in K and R were eliminated also; and that meta–rule without any markers can be considered equivalent to a simple double–pushout rule.

3. Performance in practical applications

Expression power of meta–rules depends heavily on hierarchical structure of graph models being transformed. Definition 1 places no constraints on where atoms can be nested; it would be possible, for example, to create a graph which has several nodes subordinated to an edge, which, in turn, is nested inside another edge, which is inside a node. This flexibility is a distinguishing feature of our hierarchical graphs framework, but it can be overwhelming.

In a given practical application, the models used will usually belong to some specialized subclass of hierarchical graphs. Our running example is about floor layouts; as Fig. 1 displays, nodes represent subcomponents of a house, and edges relations between these components. This means that nodes are containers and will have other nodes nested inside, in order to represent the structure of a house. So, hierarchy between nodes mirrors the hierarchy of components in the object being designed. But what about edges?

There is no obvious answer in their case the programmer who implements a floor layout design support system may make a somewhat arbitrary decision about their placement. For example, it would be possible to decide that all edges should be placed on the top level of a graph, because they don't play any role in the hierarchy of house components. Or it could be decided that each edge should have the same parent as its source node. This decision has consequences when it comes to transformation rules which modify edges – the second graph subclass is much easier to work with, because edge and at least one of the attached nodes are on the same hierarchy level.

The decision about edges influences results produced by meta–rules, too. Please recall that meta–rules were introduced in order to meet expectations of users. An average user, when presented with our running example, would say

that duplicated room should contain all subcomponents present in the original room, and that the original relations between these components (i.e. the connection between the video recorder and the TV set) should be also copied. Some percentage of users would also say that the newly created TV set should be connected to the antenna, because the old one was.

If edges representing relations are to be copied when applying the meta–rule from Fig. 3, then they must be included in $\exp(\alpha)$, which means that the "living room" node must be their parent. But the expansion set containing two nodes (VCR, TV) and two edges won't be a subgraph, and we won't be able to successfully complete the derivation algorithm. As it turns out, the optimal case has $\exp(\alpha)$ consisting of two nodes and their connecting edge. The derived rule is displayed in Fig. 4.

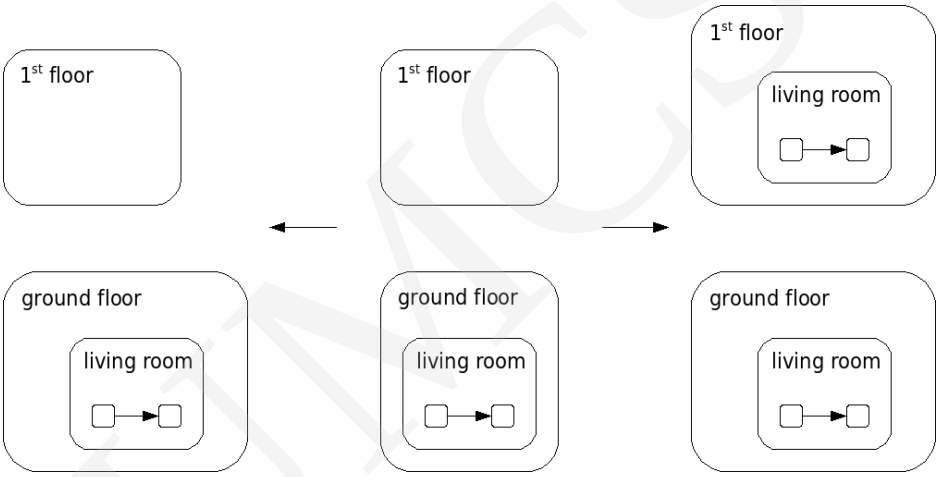

Fig. 4. A derived rule

Please note that we have managed to preserve edges representing relations internal to the expansion subgraph, and omitted edges which were crossing the boundary of $\exp(\alpha)$. It is possible to consistently obtain such results for other meta–rules and graphs if graphs used have the following properties:

- only nodes may contain subordinated atoms;
- if edge e connects nodes v and w, then $par(e)$ is equal to the closest common ancestor of v and w.

These two conditions guarantee that all expansion sets will be valid subgraphs, which means that it will be always possible to derive a rule. Edges which connect nodes in $\exp(\alpha)$ are included, those which cross the boundary of $\exp(\alpha)$ are not.

Even if derivation is guaranteed to succeed in this graph subclass, the derived rule may fail to transform the graph. It is a well–known property of double–pushout rules – they cannot be applied if they delete atoms, and those deletions would leave some edges or children dangling.

On one hand, meta–rules are more susceptible to dangling edges than simple rules; unknown node contents represented by markers may be connected to distant parts of the graph, which would prevent the rule from deleting these contents. On the other hand, simple rules have no way of dealing with dangling children, and meta-rules can use markers to represent these children and delete them en masse.

4. Conclusions

The meta–rules presented in this paper can be used to formally describe transformations operating on graph nodes along with their (unknown in advance) contents. Best results can be obtained when graphs being transformed fulfill specific requirements on atom hierarchy; in such subclass of graphs, derived rules can always be constructed, and will include all edges representing internal relations between nodes in the same marker expansion set.

The problem of duplicating external relations has not had an elegant solution so far. Also, meta–rules cannot remove nodes if it leaves some edges dangling – this is an inherent limitation of the double–pushout graph transformation approach. These two problems merit further investigation because of their importance in practical applications.

References

[1] Palacz W., *Algebraic hierarchical graph transformation*, Journal of Computer and System Sciences 68(3) (2004).

[2] Palacz W., *Hierarchical graphs in computer-aided design systems*, Ph.D. thesis, Jagiellonian University, (2006).

[3] Rozenberg G. (ed), *Handbook of graph grammars and computing by graph transformation*, Volume 1: foundations, World Scientific Publishing (1997).