Attributed Hierarchical Port Graphs and Applications

Nneka Chinelo Ene
King's College London
nneka.ene@kcl.ac.uk

Maribel Fernández
King's College London
maribel.fernandez@kcl.ac.uk

Bruno Pinaud
University of Bordeaux, LaBRI UMR CNRS 5800, France
bruno.pinaud@u-bordeaux.fr

We present attributed hierarchical port graphs (AHP) as an extension of port graphs that aims at facilitating the design of modular port graph models for complex systems. AHP consist of a number of interconnected layers, where each layer defines a port graph whose nodes may link to layers further down the hierarchy; attributes are used to store user-defined data as well as visualisation and run-time system parameters. We also generalise the notion of strategic port graph rewriting (a particular kind of graph transformation system, where port graph rewriting rules are controlled by user-defined strategies) to deal with AHP. We describe applications in two areas: functional programming and financial modelling.

1 Introduction

We present a hierarchical graph transformation system built on a new formalism, that of an *attributed hierarchical port-graph* (AHP). Port graphs [3] are graphs where edges are connected to nodes via ports. Attributed port graphs, where ports, nodes and edges carry data in the form of pairs attribute-value, have been used as a visual modelling tool in a variety of domains [4, 36, 37]. They provide a visual mechanism to describe the structure of the system under study (via a graph) together with data structuring tools (attributes represent data associated to the model, as well as visualisation and run-time system parameters). However, whereas attributed port graphs carry data of basic types, AHP permit the use of attributes of graph type in nodes. In this sense, AHP could be thought of as a "higher-order" extension of attributed port graphs. The occurrence of attributes of graph type defines a nested structure — a forest. Thus, a hierarchical port graph specifies a graph hierarchy by recursively permitting attributes of a hierarchical port graph type whilst simultaneously restricting the occurrence of non-tree like structures. This simplifies the definition of graph morphisms, which are a key notion in graph rewriting, and enforces a more natural and usable structure.

AHP forms the basis upon which graph transformation rules can be built that will enable a step-wise transformation of the graph, and unlike a regular port-graph transformation system, is able to fully capture hierarchical complexity without compromising on transparency and good visualisation. We propose the use of AHP to represent system states, and AHP rewrite rules to model the dynamic behaviour of the system. The rewriting relation generated by AHP rewrite rules can be controlled using the strategy languages already available for port graph transformation systems (see, e.g., [4, 37]) to specify rewriting positions, rule priorities, etc.

We relate our notion of hierarchical graph rewriting to the conventional rewriting of flat graphs by introducing a flattening operation, which recursively replaces each hierarchical node by its contents resulting in a standard port graph. We show that under some conditions every hierarchical graph rewriting step gives rise to a standard rewriting step on the flattened graphs by using the flattened rule.

Hierarchical graphs have been used to specify re-factoring transformations for object-oriented programs [15], to provide semantics for distributed and mobile software systems with dynamic reconfiguration [17] and in computational biology [30] to define multi-layered biochemical systems and represent molecular agents at varying levels of abstraction [13], to name but a few areas. With the aim of improving the support for graph-based programming languages, hierarchical hyper-graph transformation systems have been previously proposed, where certain hyper-edges contain hyper-graphs [14], and rewriting is defined following the double push-out approach. In this paper, we focus on port graphs, which follow the single push out approach to rewriting, and extend them by permitting nodes to carry attributes of graph type. We illustrate the use of hierarchical port graphs in two areas: securitisation (specifically, we model the secondary market "rational negligence phenomenon" [2, 23], whereby investors may choose to trade securities without performing independent evaluations of the underlying assets) and functional programming (specifically, we show how λ-terms can be defined as AHP and evaluated by port graph rewriting).

Securitisation models have been widely studied especially after the 2008 crisis (see, e.g., [27, 29]) and a port graph model illustrating the rational negligence problem is available [16]. In this paper we show how the hierarchical features of AHP can be used to refine and complete this model, catering for the more operational tiers of the securitisation process. Whilst the top tier of the rational negligence model deals with asset transfers, the operational tiers, at a basic level, encapsulate asset origination, packaging, structuring and servicing details, the key processes in the life cycle of a structured security.

Graph representations have been used in many λ-calculus evaluators to enforce sharing of computations [1]; interaction nets (a particular kind of graph rewriting formalism introduced by Lafont [26], inspired by the graphical notation of linear logic proofs) permit to implement interesting strategies of evaluation, such as optimal reduction [22]. However, in interaction net evaluators the representation of λ-abstraction is involved due to the fact that λ is a binder and explicit markers have to be used to define and manage its scope. Higher-order port graphs [20] permit to encode binders in a direct way and can be easily represented as AHP.

Summarising, our main contribution is a definition of attributed hierarchical port graph together with corresponding notions of graph morphism and rewriting relation. To highlight the suitability of attributed hierarchical port graphs as a conceptual framework we give two examples of application and compare the obtained AHP models with closely related models already available (based on standard port graphs or higher-order port graphs).

This paper is organised as follows: We recall key notions useful in our analyses in Sect. 2. Attributed Hierarchical Port Graphs are defined in Sect. 3. Applications are described in Sect. 4. Sect. 5 examines key properties of the model. Sect. 6 discusses related work. We finally conclude and briefly outline future plans in Sect. 7.

2 Background

There are many different kinds of graph transformation systems see, for instance, [7, 12, 24, 33]. In this paper we examine the transformation of port graphs [18], which have been used as a modelling tool in various domains, such as biochemistry and social networks [36, 37].

Intuitively an attributed port graph is a graph where nodes have explicit connection points, called ports, and edges are attached to ports. Nodes, ports and edges are labelled by a set of attributes describing properties such as colour, shape, etc. To formally define attributed port graphs, we follow [19], where records (i.e., sets of pairs attribute-value) are attached to graph elements.

Definition 1 (Signature) *A port graph signature ∇ consists of the following pairwise disjoint sets: $\nabla_{\mathscr{A}}$, a set of attributes; $\mathscr{X}_{\mathscr{A}}$, a set of attribute variables; $\nabla_{\mathscr{V}}$, a set of values; $\mathscr{X}_{\mathscr{V}}$, a set of value variables.*

Definition 2 (Record) *A record r over the signature ∇ is a set $\{(a_1,e_1),\ldots,(a_n,e_n)\}$ of pairs, where for $1 \leq i \leq n$, $a_i \in \nabla_{\mathscr{A}} \cup \mathscr{X}_{\mathscr{A}}$ and e_i is an expression built from $\nabla_{\mathscr{A}} \cup \nabla_{\mathscr{V}} \cup \mathscr{X}_{\mathscr{V}}$, each a_i occurs only once in r, and there is one pair where $a_i = Name$, a special element. The function Atts applies to records and returns the labels of all the attributes: $Atts(r) = \{a_1,\ldots,a_n\}$ if $r = \{(a_1,v_1),\ldots,(a_n,v_n)\}$. As usual, $r.a_i$ denotes the value v_i of the attribute a_i in r. The attribute Name identifies the record in the following sense: $\forall r_1, r_2, Atts(r1) = Atts(r2)$ if $r1.Name = r2.Name$.*

In addition to *Name*, records can contain any number of data attributes, which must be of basic type (i.e., numbers, strings, Booleans, etc.), as well as distinguished attributes *Connect*, *Attach* and *Arity*, that define structural properties of graphs, as described in Definition 3. If r is a record, $r|_D$ denotes the sub-record that excludes the structural attributes *Connect*, *Attach* and *Arity*.

Definition 3 (Port Graphs) *Let $\mathscr{N}, \mathscr{P}, \mathscr{E}$ be pairwise disjoint sets. A* port graph *over a* signature ∇ *is a tuple $G = (V,P,E,D)_{\mathscr{L}}$ where $V \subseteq \mathscr{N}$ is a finite set of nodes ($n, n_1, \ldots$ range over nodes); $P \subseteq \mathscr{P}$ is a finite set of ports ($p, p_1, \ldots$ range over ports); $E \subseteq \mathscr{E}$ is a finite set of edges between ports ($e, e_1, \ldots$ range over edges; edges are undirected and two ports may be connected by more than one edge); D is a set of records over ∇, and $\mathscr{L} : V \cup P \cup E \to D$ is a labelling function such that:*

- *for each edge $e \in E$, $\mathscr{L}(e)$ contains an attribute Connect whose value is the pair $\{p_1, p_2\}$ of ports connected by e.*
- *for each port $p \in P$, $\mathscr{L}(p)$ contains an attribute Arity whose value is the number of edges connected to this port, and an attribute Attach whose value is the node to which p belongs.*
- *For each node $n \in V$, $\mathscr{L}(n)$ contains an attribute Interface whose value is the set of names of the ports attached to n, that is, $\mathscr{L}(n).Interface = \{\mathscr{L}(p_i).Name \mid \mathscr{L}(p_i).Attach = n\}$, satisfying the following constraint: $\mathscr{L}(n_1).Name = \mathscr{L}(n_2).Name \Rightarrow \mathscr{L}(n_1).Interface = \mathscr{L}(n_2).Interface$.*

A port graph rewriting rule $L \Rightarrow_C R$ can itself be seen as a port graph consisting of two port graphs L and R together with an "arrow" node linked to L and R by a set of edges that specify a mapping between ports in L and R (see Figure 3 for an example of a rule, the edges involving the arrow node are red in the figure). The pattern, L, is used to identify sub-graphs in a given graph which should be replaced by an instance of the right-hand side, R, provided the condition C holds. The red edges indicate how the instance of R should be linked to the remaining part of the graph. Each of the ports attached to the arrow node has an attribute Type $\in \nabla_{\mathscr{A}}$, which can have three different values: bridge, wire and black-hole, values that indicate how a rewriting step using this rule should affect the edges that connect the redex to the rest of the graph and that satisfy the following conditions: A port of type bridge must have edges connecting it to L and to R (one edge to L and one or more to R); a port of type black-hole must have edges connecting it only to L (at least one edge); a port of type wire must have exactly two edges connecting to L and no edge connecting to R.

Intuitively, a port of type bridge in the arrow node connecting to p_1 in L and p_2 in R indicates that p_1 survives the reduction and becomes p_2. A port in L connected to a black-hole port in the arrow node does not survive the reduction; all edges connected to this port in the graph are deleted when the reduction step takes place. A port of type wire connected to two ports p_1 and p_2 in the left-hand side triggers a particular rewiring, which takes all the ports that are connected to (the image of) p_1 in the redex and creates an edge for each of those ports to each of the ports connected to (the image of) p_2. We refer to [18, 19] for more details and examples.

To define the rewrite relation, we use port graph morphisms, which preserve the graph structure and the values of the attributes (instantiating variables that occur in patterns).

Definition 4 (Match) *Let $L \Rightarrow_C R$ be a port graph rewrite rule and G a port graph. We say a match $g(L)$ of L (i.e., a redex) is found in G if there is a port graph morphism g from L to G (hence $g(L)$ is a sub-graph of G), C holds in $g(L)$, and for each port in L that is not connected to the arrow node, its corresponding port in $g(L)$ is not an extremity in the set of edges of $G - g(L)$.*

Let G be a port graph. A *rewrite step $G \Rightarrow H$* via the port graph rewrite rule $L \Rightarrow_C R$ is obtained by replacing in G a match $g(L)$ by $g(R)$ and redirecting the edges incident to ports in $g(L)$ to ports in $g(R)$ as indicated by the arrow node. The last point in Definition 4 ensures that ports in L that are not connected to the arrow node are mapped to ports in $g(L)$ that have no edges connecting them with ports outside the redex, thus ensuring that there will be no dangling edges when $g(L)$ is replaced by $g(R)$.

A sequence of rewriting steps is called a *derivation*. A *derivation tree* is a collection of derivations with a common root.

PORGY [18] is an interactive environement that includes functionalities to create port graphs and port graph rewrite rules, and to apply rules to graphs (according to user-defined strategies). It provides a visual representation of the derivation tree, which can be used to analyse the rewriting system; however, it lacks mechanisms to define graphs in a modular way. To facilitate the design of modular port graph models, we plan to extend this tool with attributed hierarchical graphs (presented in the next section).

3 Attributed Hierarchical Port-graphs

We extend the notion of a port graph to support a multi-level structure. The key idea is the introduction of a new kind of attribute in the signature, which we call *Ladder*, whose value is of type graph. In the definition of AHP we use the concept of *Interface* of a graph, which is the set of free ports in the graph (i.e., ports that do not have edges attached to them).

Definition 5 (AHP-Signature) *An AHP-signature $\nabla_{\mathscr{G}}$ consists of a port graph signature (see Definition 1), where the set of attribute names $\nabla_{\mathscr{A}}$ includes a distinguished element* Ladder*, and the sets of values, $\nabla_{\mathscr{V}}$, and value variables, $\mathscr{X}_{\mathscr{V}}$, include elements of type graph.*

As in the case of port graph signatures, we consider records over $\nabla_{\mathscr{G}}$ and distinguish structural attributes and data attributes. The structural attributes are *Connect*, *Attach*, *Arity* and *Ladder*. The data attributes are in turn split into basic and graph attributes. We denote by $\nabla_{\mathscr{V}}^{G}$ (resp., $\mathscr{X}_{\mathscr{V}}^{G}$) the subset of $\nabla_{\mathscr{V}}$ consisting of values of type graph (resp., the subset of $\mathscr{X}_{\mathscr{V}}$ consisting of variables of type graph), and by $\nabla_{\mathscr{V}}^{B}$ (resp., $\mathscr{X}_{\mathscr{V}}^{B}$) the rest of the data elements. Thus, the elements $G_1, G_2, \ldots$ of $\nabla_{\mathscr{V}}^{G}$ are attributed port graphs and the elements $\mathfrak{X}_1, \mathfrak{X}_2, \ldots$ of $\mathscr{X}_{\mathscr{V}}^{G}$ are variables representing unknown port graphs, each with an associated interface (list of port names) denoted by *Interface*$(\mathfrak{X})$. AHP with variables in Ladder attributes will be used only in rewrite rules.

The set of attributed hierarchical port graphs will be defined by induction, relying on the fact that graph values in records of a given graph are previously defined graphs. More precisely, AHP will be defined by levels, where the graphs at level i may use graphs of a lower level only.

The levels induce a notion of hierarchy in the graph: in an AHP, the nesting of graph-type attributes defines a tree structure, i.e., there are no cyclical connections between graph attributes, that is, no edges linking graphs at different levels, or graphs at peer level. This reduces the complexity of the matching problem, ensures compositionality (which is important when defining rewriting and flattening relations:

components of the graph can be replaced and the resulting overall graph is equivalent to the original graph, whereas it is not obvious how to replace components if there are edges across components), and facilitates the visualisation of the graph, generating a greater ease in user-understanding.

Definition 6 (Attributed Hierarchical Port-graph) *An AHP (*Attributed Hierarchical Port-graph*) over a signature $\nabla_{\mathscr{G}}$ is an element of the set $\mathscr{H} = \bigcup_{i \geq 0} \mathscr{H}_i$, where $\mathscr{H}_i$, the set of AHP at* level *i, is defined as follows:*

An AHP at level 0 is an attributed port graph as specified in Definition 3 (i.e., without graph-type attributes).

An AHP at level $i+1$ is a tuple $(V,P,E,D)_{\mathscr{L}}$, of a set V of nodes, a set P of ports, a set E of edges, and a set D of records over $\nabla_{\mathscr{G}}$, together with a labelling function $\mathscr{L}$ that associates records in D to elements in V,P,E, such that only nodes may be labelled with records containing the attribute Ladder, and the value of any Ladder attribute in $\mathscr{L}(V)$ is an AHP at level $j \leq i$ or a variable of type graph ($\mathfrak{X} \in \mathscr{X}^G_{\mathscr{V}}$). The Ladder graph must have the same interface as the node, i.e., the Ladder graph should have same number of free ports, with the same names, as the node: Interface($\mathscr{L}(n)$.Ladder) $= \mathscr{L}(n)$.Interface.

Given an AHP G, we assume that the graphs in Ladder attributes within G are pairwise disjoint (that is, they do not share nodes, ports or edges, and therefore there are no edges across graphs at different levels in G, or graphs of the same level in different nodes).

Figure 1 shows an example of an AHP graph: the node *A* in the top level has a Ladder attribute containing the graph shown in the centre of the figure, which in turn contains a node *Pools* with a Ladder attribute (the graph in the rightmost part of the figure).

More flexible notions of hierarchical graphs, where edges between nodes at the same level in the hierarchy are permitted, will be considered in future work.

To define the rewriting relation, we need AHP-morphisms. The definition of AHP-morphism (Definition 7) ensures that corresponding data attributes in *G* and *H* have the same values (except at positions where there are variables in *G*, which are instantiated in *H*), and attachment of ports, edge connections and ladder graphs are preserved. If f is a morphism from *G* to *H*, we will denote by $f(G)$ the sub-graph of *H* consisting of the set of nodes, ports, edges and records that are images of nodes, ports, edges and records in *G*.

Definition 7 (AHP Morphism) *Given two AHP $G = (V_G,P_G,E_G,D_G)_{\mathscr{L}_G}$ and $H = (V_H,P_H,E_H,D_H)_{\mathscr{L}_H}$ over the same signature $\nabla_{\mathscr{G}}$, a* morphism *f from G to H, denoted $f : G \rightarrow H$, is a family of* injective functions *$\langle f_V : V_G \rightarrow V_H, f_P : P_G \rightarrow P_H, f_E : E_G \rightarrow E_H \rangle$ and instantiation functions $f_1 : \mathscr{X}_{\mathscr{A}} \rightarrow \nabla_{\mathscr{A}}$, $f_2 : \mathscr{X}_{\mathscr{V}} \rightarrow \nabla_{\mathscr{V}}$ (which induce an instantiation function on records) such that*

- *$f_P : P_G \rightarrow P_H$ is a mapping from the set of ports of G to the set of ports of H such that if $p \in P_G$ then $f_1(f_2(\mathscr{L}_G(p))) = \mathscr{L}_H(f_P(p))$.*
- *$f_V : V_G \rightarrow V_H$ is a mapping from the set of nodes of G to the set of nodes of H such that if $n \in V_G$ then $f_1(f_2(\mathscr{L}_G(n)))|_D = \mathscr{L}_H(f_V(n))|_D$ (i.e., the morphism preserves data-attribute values), if $n = \mathscr{L}_G(p)$.Attach for some $p \in P_G$, then $f_V(n) = \mathscr{L}_H(f_P(p))$.Attach (i.e., the morphism preserves the attachment of ports to nodes), and such that $f(\mathscr{L}_G(n).Ladder) = \mathscr{L}_H(f_V(n)).Ladder$.*
- *$f_E : E_G \rightarrow E_H$ is a mapping from the set of edges of G to the set of edges of H such that if $e \in E_G$ then $f_1(f_2(\mathscr{L}_G(e)))|_D = \mathscr{L}_H(f_E(e))|_D$ and $\overline{f_P}(\mathscr{L}_G(e).Connect) = \mathscr{L}_H(f_E(e)).Connect$ where $\overline{f_P}$ denotes the extension of the function f_P to sets of ports (i.e., the morphism preserves the edge connections).*

As in the case of standard port graphs, when using morphisms to define rewriting, we will only allow the use of variable labels on one of the graphs: the graph L on the left-hand side may include variable labels, whilst the graph to be rewritten may not.

Definition 8 (AHP Rewrite Rule) *An AHP rewrite rule $L \Rightarrow_C R$ is an AHP graph that consists of two sub-graphs L and R together with a node (called arrow node) that may carry a condition C and whose edges link to ports in the top level of L and in the top level of R, capturing the correspondence between top level ports in L and R. Each variable in $\mathscr{X}_G$ occurs at most once in L. The three types of ports in the arrow node (i.e. bridge, wire and black-hole) remain the same as for conventional port graphs.*

Figure 2 is an example of an AHP rule.

Definition 9 (AHP Match) *Let $L \Rightarrow_C R$ be an AHP rewrite rule and G an AHP graph. A match between L and G is said to take place if an AHP morphism g between L and a sub-graph of G can be identified, such that $g(L)$ satisfies C and for each port in L that is not connected to the arrow node, its corresponding port in $g(L)$ is not an extremity in the set of edges of $G - g(L)$.*

Note that to find a match, where an attribute is of type graph, a new morphism is sought (recursively) between the graph attribute in L and the value of the Ladder attribute in the corresponding node in G.

The rewriting relation generated by AHP rules, denoted $G \Rightarrow H$, is defined in a similar way as for attributed port graphs (see Section 2): a subgraph $g(L)$ in G is replaced by $g(R)$ and the edges incident to $g(L)$ are rewired as indicated by the arrow node, provided the morphism g satisfies the matching conditions. The restriction to linear occurrences of graph-type variables in L ensures that the morphism always yields a well-defined transformation. Since a rule is an AHP, the arrow node can only link ports in L and R at the top level (no edges can cross level boundaries) and the replacement of a subgraph by another one maintains the tree structure. As a consequence, there are no dangling edges when $g(L)$ is replaced with $g(R)$ in an AHP-rewriting step. We study properties of the rewriting relation in Section 5.

4 Applications

We briefly describe two case studies: Asset-Backed Securitisation and the λ-calculus.

The ABS-AHP Securitisation Model. As defined in [28] "Securitisation is the process of converting cash flows arising from underlying assets or debts/receivables (typically illiquid such as corporate loans, mortgages, car loans and credit cards receivables) due to the originator into a smoothed liquid marketable repayment stream" and this ensures that the originator can raise asset-backed finance through loans or the issuance of debt securities. An *originator* is any financial intermediary with a portfolio of assets on its balance sheet. *Assets* represent loans to clients or *obligors* who make regular installment payments to the originator to clear their debts. In a securitisation, assets are selected, pooled and transferred to a tax neutral, bankruptcy-remote, liquidation-efficient (i.e bankruptcy avoiding), *special purpose entity* (SPE) or *special purpose vehicle* (SPV), who funds them by issuing *securities*. In general, an ABS (asset-backed security), or simply asset if there is no ambiguity, is any securitisation issue backed by consumer loans, car loans, credit cards, etc.

ABSs are *rated* by *credit agencies* (e.g., Standard and Poor's) and are bought by *investors* (usually banks or other financial agents), who are paid back over time by the SPV from the stream of receivables. Investors can then trade these ABSs, characterised by varying levels of liquidity based on the type of asset, with others in the secondary market.

The ABS trading model described in [2] has been specified as a port graph rewriting system and implemented in PORGY [16]; in this paper we incorporate lower, more operational tiers into the model, by means of attributed hierarchical port graphs. The operational tiers will lead to the computation of the security's "pay off" attribute at any point in time (in this case, the profit made from successfully reselling the security, given varying toxicity likelihood values and the internal examination of the time-value-of-money).

We represent the full ABS universe hierarchically and enforce information flow bidirectionally. Hierarchical port graph rewriting rules and strategies control the step-wise evolution of the graphs. The derivation tree is used for plotting and analysing parameters. The asset trading model [16] sits at the top level of the model hierarchy and below this system lie several more deterministic subsystems that model encapsulate asset origination, packaging, structuring and servicing processes, and therefore aid in enforcing internal checks as a result of complete system integration. Figure 1 summarises all the major components of a securitisation as currently recorded in the system. The first, second and third graphs respectively represent the Secondary Market, Structuring and Origination tiers respectively. B is an agent, the buyer or seller bank (in the actual system there are eleven agents at the secondary market level), A is an asset or asset-backed security, Tranches represents the internal structuring or content of an asset and Pools the connection to underlying that provide an income stream.

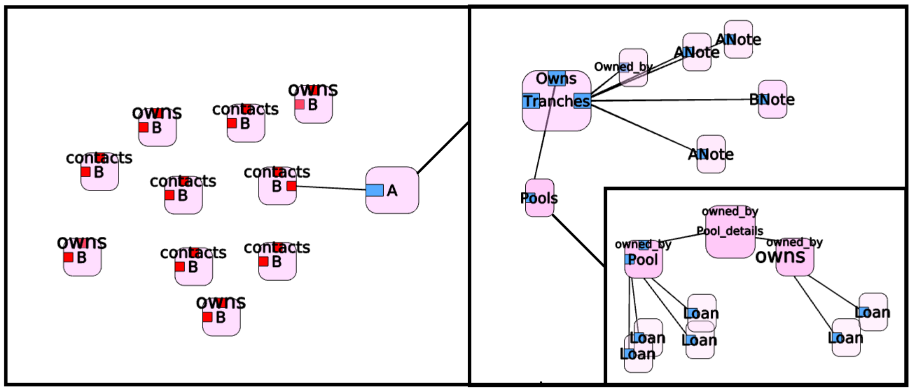

Figure 1: Sample Starting Graph

The hierarchical rewrite rules that drive execution have patterns that cut across the two or three identified tiers. The transformation of the context graph shall be seen in asset transfers, general node movements, and colour changes that indicate changes of state. Core structuring and origination processes may be better carried out from an external program and transferred into the graph platform (PORGY can import graphs through a GXL plug-in). Rules such as that which aid in the structuring of the asset as seen in Figure 3 or its hierarchical version as seen in Figure 2, can then be efficiently done from within the system, fulfilling the original objectives of a system that facilitates transparency. A flattening algorithm that permits same-level interactions as seen in the sample rule is described in the next section.

Compared with standard port graph models, where all the different features and processes have to be represented in the same "flat" graph, the hierarchical model facilitates the analysis of the system as it is possible to hide the non-relevant details in lower levels to focus on the features of interest.

Lambda-terms: An HOPG-AHP Encoding. The λ-calculus [8] is a paradigmatic model of functional computation. We now consider common λ-term representations, and investigate AHP encodings.

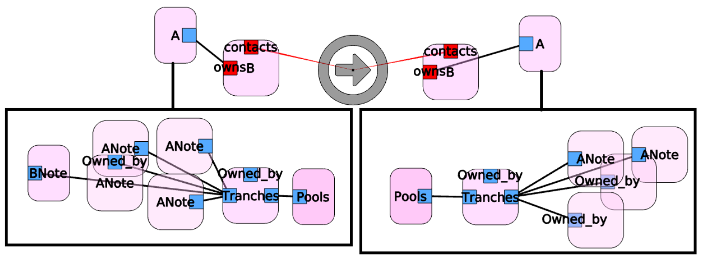

Figure 2: sample Hierarchical Rewrite Rule: “Update Ladder”

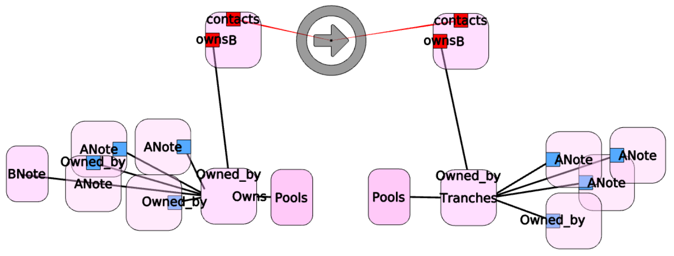

Figure 3: FLattened Version of Sample Hierarchical Rewrite Rule: “Update Ladder”

Intuitionistic logic proofs (which, by the Curry-Howard isomorphism are equivalent to λ-terms) expressed in a natural deduction style can be inductively translated into conventional port graphs (standard interaction net translations can be used, since interaction nets are a particular kind of port graph). This representation works well for some aspects of logic but not where boxes are required as this introduces a two level structure that is not available in standard port graphs or interaction nets. Boxes are represented by extra nodes requiring additional rules for book-keeping.

In [1], a representation of intuitionistic logic proofs (or λ-terms) is given by means of *higher-order port-graphs* (HOPG). Port-graphs coupled with higher-order capabilities overcome some of the drawbacks of interaction net representations. HOPG [20] extend port graphs with *higher-order variable nodes*, which can be instantiated by port graphs. A labelled higher-order port-graph consists of a collection of first-order nodes (whose names can be constants or variables), a collection of higher-order nodes (whose names are variables), a collection of ports attached to nodes (higher-order variable nodes contain only variable ports), undirected edges, and labelling functions that determine the concrete properties (attributes and values) of each graph element.

The notion of port graph morphism has been extended in [20] to take into account higher-order nodes; [20] proposes a matching algorithm for HOPGs, which caters to the fact that higher order variable nodes cannot be handled by standard (first order) matching.

AHP graphs subsume HOPGs by introducing an abstraction level that not only fulfils the original HOPG objective of simulating “boxes” or the grouping of a collection of nodes within one node, all interfaces matching, but that also maintains a tree structure that can be recursively flattened on demand

Algorithm 1: Flattening Function $\mathscr{T}$

```
1 Let $G$ be an AHP graph without variables of type graph.
2 $\mathscr{T}(G) = G$     if $G \in \mathscr{H}_0$
3 $\mathscr{T}(G) = G'$    otherwise, where $G'$ is obtained from $G$ by replacing each hierarchical node $n$
4                  in $G$ with $\mathscr{T}(\mathscr{L}(n).Ladder)$, connecting edges incident to ports in $n$ to the
5                  corresponding ports in $\mathscr{T}(\mathscr{L}(n).Ladder)$
```

to produce a conventional graph. An AHP can act as an HOPG and also permits the nesting of additional nodes. Similar to the example HOPG-implementation given in [20], an AHP graph can directly represent a proof (or a λ-term) that contains a box by using a hierarchical node *Box* whose Ladder attribute contains the box structure. A corresponding sample AHP rule that can act on the graph could represent one step in the cut-elimination procedure that seeks to normalise the proof, as shown in [20]. However, instead of giving a direct encoding of λ-terms into AHP graphs, we describe below an encoding of HOPG in AHP graphs, which shows AHP graphs capture and also extend the higher-order capabilities introduced in [20] by providing a multi-level representation.

The key idea to encode an HOPG as an AHP graph is to simulate higher-order variables by introducing an abstraction level: more precisely, a higher-order variable node labelled by $\mathfrak{X}$ in the HOPG is represented by a node with a Ladder attribute with value $\mathfrak{X}$ in the corresponding AHP graph. In other words, where a higher-order variable is used in an HOPG to represent an unknown subgraph, a node with a Ladder attribute containing a graph variable is used. The parent node can be seen as a place-holder, named appropriately. It maintains an interface that the sub-graph, when instantiated, will also adopt.

5 Properties

We start with a soundness property: rewriting an AHP produces another AHP, that is, rewriting does not leave dangling edges and does not break the hierarchical tree-structure. This is because a rule is itself an AHP and therefore the arrow node (which describes the rewiring of the edges) cannot introduce edges across different level subgraphs.

Property 1 *Let G be an AHP and $L \Rightarrow_C R$ an AHP rewrite rule. If $G \Rightarrow H$ using $L \Rightarrow_C R$ then H is an AHP.*

In order to relate our notion of AHP rewriting to the conventional transformation of flat port graphs, and to test our example model in the current version of PORGY, we have implemented a flattening function $\mathscr{T}$ that unfolds an attributed hierarchical port graph without graph-type variables into a regular port graph as seen in Algorithm 1. Since the graph in the Ladder attribute of a node n AHP has the same interface as n, we can flatten the graph by replacing each hierarchical node with a flattened version of its Ladder graph (recursively), redirecting the edges incident to n to the corresponding ports in the flattened Ladder.

It is easy to see that the recursive definition of the flattening function ensures the result is always a regular port graph (i.e, an AHP at level 0). Moreover, a hierarchical rewriting step induces a corresponding ”flat” rewriting step (the converse does not hold since structural information gets lost in the flattening process).

Property 2 *Assuming there are no graph-type variables:*

1. *If G is an AHP, then $\mathscr{T}(G)$ is a regular port graph. Similarly, the flattening of an AHP-rewrite rule produces a regular port graph rule.*

2. *If G is an AHP and $L \Rightarrow_C R$ an AHP rewrite rule, such that $G \Rightarrow H$ using $L \Rightarrow_C R$, then $\mathscr{T}(G) \Rightarrow \mathscr{T}(H)$ using $\mathscr{T}(L \Rightarrow_C R)$.*

6 Related Work

Various extensions of graph formalisms have been previously defined with the aim to provide abstraction and structuring features in graph-based modelling tools. A prominent example is the concept of bigraph introduced by Milner (see [9, 31]) to model computation on a global scale. Bigraphs are graphs whose nodes may be nested, thus providing direct support to notions of locality (nesting of nodes) and connectivity (edges), which are key aspects of mobile systems [32]. Formally, bigraphs are defined in terms of two structures that share the same node set: place graphs and link graphs. A place graph is a forest (and in this sense bigraphs define a notion of hierarchy similar to AHP's), whereas the link graph is a hyper-graph. Closely related to bigraphs are the deduction graphs proposed by Geuvers and Loeb [21], able to intuitively and graphically represent proofs.

Hierarchical hyper-graphs [14] also provide structuring features: attributes of type graph are permitted within hierarchical hyper-graphs in association with edges as "frames" where frames are hyper-edges that can contain hierarchical hyper-graphs of an arbitrary nesting depth and, also in a similar fashion to our solution, edges cutting across components are prohibited. Similar to our proposal, a recursive flattening approach is highlighted, albeit via hyper-edge replacement as opposed to node replacement.

Bigraphs and hierarchical hyper-graphs are equipped with a formal, categorical semantics. Where a double push-out graph transformation approach applies naturally to the transformation of hierarchical hyper-graphs, the dynamic theory of bigraphs relies on a notion of relative push-out. For AHP graphs, we follow the single push-out approach advocated in previous port graph transformation systems. However, the notion of matching we define in this paper is purely operational, and can be easily implemented as an extension of existing port graph matching algorithms.

H-Graphs [10] model hierarchy by labelling nodes of set N over a set of atoms A or permitting an embedded directed sub-graph created from N. H-Graphs in practice are used to model run-time data structures for the definition of programming language semantics and H-Graph grammars model operations over these structures by substituting an atomic node with a H-Graph. A benchmark developed in 2001 [10] analyses hierarchical graphs in terms of underlying graph structure, the nested packages and a coupling mechanism, investigating the two major approaches presented by H-graph grammars and hierarchical hyper-graphs, and providing a means by which to compare their properties uniformly.

Distributed hierarchical graphs form the basis upon which the agent-based OWL semantic web ontological model is built and in which graph transformations can take place at various levels of abstraction [35]. There is however no clear handling of structural connectivity by the use of ports and limited attribute handling. Other interesting hierarchical representations and implementations include that of L-Graphs in which edges and nodes are labelled by graphs [34], that of the verbose property graphs used in graph analytics [25], and [38]; also using attributes and multi-typed sub-graphs.

Specifically for port graphs, an abstract higher-order calculus inspired by the ρ-calculus [11] is defined in [5, 6], where terms can refer to objects that are port-graphs and variables may range over rewrite rules. This calculus can be seen as a combination of first-order rewrite rules with abstraction, but the notion of graph morphism (which is the basis for the matching and the rewriting algorithm) is first-order. As already mentioned, the higher-order port graphs found in [20] include a class of nodes labelled by variables that can be instantiated by graphs, thus offering abstraction but no additional structuring capabilities.

7 Conclusions and Future Work

We have introduced AHP as a means to specify hierarchical models in a manner that is concise, visual, transparent, modular and feasible. Future work will include a full implementation of this design within PORGY and completing the construction of the case-studies outlined. Despite additional computational complexity in matching, our strict hierarchical tree structure could be extended to permit more flexibility and edges linking nodes at different levels. Boundary-crossing edges are permitted within some of the visual languages used in software modelling (e.g., to represent UML diagrams, given inheritance dependencies between separate sub-graphs).

References

[1] S. Alves, M. Fernández, and I. Mackie. A new graphical calculus of proofs. In *Proceedings 6th International Workshop on Computing with Terms and Graphs, TERMGRAPH 2011, Saarbrücken, Germany, 2nd April 2011.*, pages 69–84, 2011.

[2] K. Anand, A. Kirman, and M. Marsili. Epidemics of rules, rational negligence and market crashes. *The European Journal of Finance*, 19(5):438–447, 2013.

[3] O. Andrei. *A Rewriting Calculus for Graphs: Applications to Biology and Autonomous Systems. (Un calcul de réécriture de graphes : applications à la biologie et aux systèmes autonomes).* PhD thesis, National Polytechnic Institute of Lorraine, Nancy, France, 2008.

[4] O. Andrei, M. Fernández, H. Kirchner, G. Melançon, O. Namet, and B. Pinaud. Porgy: Strategy-driven interactive transformation of graphs. In *TERMGRAPH*, pages 54–68, 2011.

[5] O. Andrei and H. Kirchner. A rewriting calculus for multigraphs with ports. *Electr. Notes Theor. Comput. Sci.*, 219:67–82, 2008.

[6] O. Andrei and H. Kirchner. A higher-order graph calculus for autonomic computing. In M. Lipshteyn, V. E. Levit, and R. M. McConnell, editors, *Graph Theory, Computational Intelligence and Thought*, pages 15–26. Springer-Verlag, Berlin, Heidelberg, 2009.

[7] H. Barendregt, M. van Eekelen, J. Glauert, J. R. Kennaway, M. Plasmeijer, and M. Sleep. Term graph rewriting. In *Proceedings of PARLE, Parallel Architectures and Languages Europe*, number 259-II in LNCS, pages 141–158, Eindhoven, The Netherlands, 1987. Springer-Verlag.

[8] H. P. Barendregt. Functional programming and lambda calculus. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 321–363. 1990.

[9] L. Birkedal, T. C. Damgaard, A. J. Glenstrup, and R. Milner. Matching of bigraphs. *Electr. Notes Theor. Comput. Sci.*, 175(4):3–19, 2007.

[10] G. Busatto and B. Hoffmann. Comparing notions of hierarchical graph transformation. *Electr. Notes Theor. Comput. Sci.*, 50(3):310–317, 2001.

[11] H. Cirstea and C. Kirchner. The rewriting calculus — Part I *and* II. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, May 2001.

[12] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation - part i: Basic concepts and double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 163–246. World Scientific, 1997.

[13] V. Danos, J. Feret, W. Fontana, R. Harmer, J. Hayman, J. Krivine, C. Thompson-walsh, and G. Winskel. Graphs, rewriting and causality in rule-based models.

[14] F. Drewes, B. Hoffmann, and D. Plump. Hierarchical graph transformation. *Journal of Computer and System Sciences*, 64(2):249 – 283, 2002.

[15] N. V. Eetvelde and D. Janssens. A hierarchical program representation for refactoring. *Electr. Notes Theor. Comput. Sci.*, 82(7):91–104, 2003.

[16] N. Ene, M. Fernández, and B. Pinaud. Graph models for capital markets. Available from https://nms.kcl.ac.uk/nneka.ene/papers.html.

[17] G. Engels and R. Heckel. *Graph Transformation as a Conceptual and Formal Framework for System Modeling and Model Evolution*, pages 127–150. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[18] M. Fernández, H. Kirchner, and B. Pinaud. Strategic port graph rewriting: An interactive modelling and analysis framework. In *Proceedings 3rd Workshop on GRAPH Inspection and Traversal Engineering, GRAPHITE 2014, Grenoble, France, 5th April 2014.*, pages 15–29, San Diego, CA, USA, 2014. Society for Computer Simulation International.

[19] M. Fernández, H. Kirchner, and B. Pinaud. Strategic Port Graph Rewriting: an Interactive Modelling Framework. Research report, Inria ; LaBRI - Laboratoire Bordelais de Recherche en Informatique ; King's College London, 2017.

[20] M. Fernández and S. Maulat. Higher-order port-graph rewriting. In S. Alves and I. Mackie, editors, *Proceedings 2nd International Workshop on Linearity, LINEARITY 2012, Tallinn, Estonia, 1 April 2012.*, volume 101 of *EPTCS*, pages 25–37, 2012.

[21] H. Geuvers and I. Loeb. Natural deduction via graphs: formal definition and computation rules. *Mathematical Structures in Computer Science*, 17(3):485–526, 2007.

[22] G. Gonthier, M. Abadi, and J. Lévy. The geometry of optimal lambda reduction. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Albuquerque, New Mexico, USA, January 19-22, 1992*, pages 15–26, 1992.

[23] G. Gorton and A. Metrick. Securitization. Working Paper 18611, National Bureau of Economic Research, December 2012.

[24] A. Habel, J. Müller, and D. Plump. Double-pushout graph transformation revisited. *Mathematical Structures in Computer Science*, 11(5):637–688, 2001.

[25] M. Junghanns, A. Petermann, and E. Rahm. Distributed grouping of property graphs with gradoop. In *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 6.-10. März 2017, Stuttgart, Germany, Proceedings*, pages 103–122, 2017.

[26] Y. Lafont. Interaction nets. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*, pages 95–108, 1990.

[27] S. Markose. Systemic risk analytics: A data-driven multi-agent financial network (mafn) approach. *Journal of Banking Regulation*, 14(3-4):285–305, 2013.

[28] S. Markose, Y. Dong, and B. Oluwasegun. An multi-agent model of rmbs, credit risk transfer in banks and financial stability: Implications of the subprime crisis, 2008.

[29] S. M. Markose, B. Oluwasegun, and S. Giansante. Multi-agent financial network (mafn) model of US collateralized debt obligations (CDO):regulatory capital arbitrage, negative CDS carry trade, and systemic risk analysis. In *Banking, Finance, and Accounting*, pages 561–590. IGI Global, Hershey, U. S. A., July 2014.

[30] O. Mason and M. Verwoerd. Graph Theory and Networks in Biology. *eprint arXiv:q-bio/0604006*, Apr. 2006.

[31] R. Milner. Bigraphical reactive systems: basic theory. Technical Report UCAM-CL-TR-523, University of Cambridge, Computer Laboratory, Sept. 2001.

[32] R. Milner. Pure bigraphs: Structure and dynamics. *Inf. Comput.*, 204(1):60–122, 2006.

[33] D. Plump. Term graph rewriting. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2: Applications, Languages, and Tools*, pages 3–61. World Scientific, 1998.

[34] H. Schneider. On categorical graph grammars integrating structural transformations and operations on labels. *Theoretical Computer Science*, 109(1):257 – 274, 1993.

[35] A. Shaban–Nejad and V. Haarslev. Managing changes in distributed biomedical ontologies using hierarchical distributed graph transformation. *International Journal of Data Mining and Bioinformatics*, 11(1):53–83, 2015.

[36] A. M. Smith, W. Xu, Y. Sun, J. R. Faeder, and G. Marai. Rulebender: integrated modeling, simulation and visualization for rule-based intracellular biochemistry. *BMC Bioinformatics*, 13(8), 2012.

[37] J. Vallet, H. Kirchner, B. Pinaud, and G. Melançon. A visual analytics approach to compare propagation models in social networks. In A. Rensink and E. Zambon, editors, *Proc. Graphs as Models, GaM 2015*, volume 181 of *EPTCS*, pages 65–79, 2015.

[38] G. Ślusarczyk, A. Łachwa, W. Palacz, B. Strug, A. Paszyńska, and E. Grabska. An extended hierarchical graph-based building model for design and engineering problems. *Automation in Construction*, 74:95 – 102, 2017.