HAL
open science

Attributed Hierarchical Port Graphs and Applications

Nneka Chinelo Ene, Maribel Fernández, Bruno Pinaud

To cite this version:

Nneka Chinelo Ene, Maribel Fernández, Bruno Pinaud. Attributed Hierarchical Port Graphs and Applications. 4th International Workshop on Rewriting Techniques for Program Transformations and Evaluation (WPTE 2017), Sep 2017, Oxford, United Kingdom. pp.2-19, 10.4204/EPTCS.265.2 . hal-01721101

HAL Id: hal-01721101

https://hal.science/hal-01721101v1

Submitted on 1 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Attributed Hierarchical Port Graphs and Applications

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

Maribel Fernández
King's College London
Dept. of Informatics
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 following the Single Push-out approach. We outline examples of application 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 [4] 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 (i.e., records), have been used as a visual modelling tool in a variety of domains [5, 43, 44]. 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 are flat structures, AHP are hierarchical structures that permit the nesting of graphs: nodes may have "ladders" to other graphs down the hierarchy. From the theoretical point of view, ladders are defined as additional structural components in the graph. From a practical point of view, this can be implemented by using attributes of *graph type* in records associated with 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 in nodes defines a forest structure (a hierarchy, where the graphs at level 0 are standard port graphs, and the nodes of graphs at level $i+1$ can include a ladder to a graph at level i). Formally, a hierarchical port graph is defined inductively using standard port graphs as base case. The restriction to a hierarchy of graphs (as opposed to arbitrary "graphs of graphs") simplifies the definition of AHP 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. Unlike a regular port-graph transformation system, AHP rules 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., [5, 44]) 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

H. Cirstea, D. Sabel (Eds.): Fourth International Workshop on Rewriting Techniques for Program Transformations and Evaluation (WPTE 2017)
EPTCS 265, 2018, pp. 2–19, doi:10.4204/EPTCS.265.2

© N. Ene, M. Fernández, & B. Pinaud
This work is licensed under the Creative Commons Attribution License.

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 [17], to provide semantics for distributed and mobile software systems with dynamic reconfiguration [20] and in computational biology [35] to define multi-layered biochemical systems and represent molecular agents at varying levels of abstraction [15], 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 [16], and rewriting is defined following the double push-out approach. In this paper, we focus on port graphs, which follow the single push-out (SPO) approach to rewriting, and provide an SPO semantics for AHP.

We illustrate the use of hierarchical port graphs in two areas: securitisation (specifically, we model the secondary market "rational negligence phenomenon" [3, 26], whereby investors may choose to trade securities without performing independent evaluations of the underlying assets) and functional programming (specifically, λ-term evaluation).

Securitisation models have been widely studied especially after the 2008 crisis (see, e.g., [32, 34]) and a port graph model illustrating the rational negligence problem is available [19]. 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; interaction nets (a particular kind of graph rewriting formalism introduced by Lafont [29], inspired by the graphical notation of linear logic proofs) permit to implement interesting strategies of evaluation, such as optimal reduction [25]. 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 [23] permit to encode binders in a direct way and can be easily represented as AHP. In future work we will explore the links between hierarchical and higher-order port graphs.

Summarising, our main contribution is a definition of attributed hierarchical port graph, together with corresponding notions of graph morphism and rewriting relation, following the SPO approach. 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. Examples are described in Sect. 4. Sect. 5 examines key properties of AHP graphs. 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, [14, 27, 40, 40]. In this paper we examine the transformation of port graphs [21], which have been used as a modelling tool in various domains, such as biochemistry and social networks [43, 44].

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 [21], 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 as first component of a pair 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: $\forall r_1,r_2, Atts(r_1) = Atts(r_2)$ if $r_1.Name = r_2.Name$.*

In addition to *Name*, records may contain any number of data attributes, which must be of basic type (i.e., numbers, strings, Booleans, etc.).

Definition 3 (Port Graph) *A* port graph *over a* signature ∇ *is a tuple $G = (V,P,E,D)_{\mathscr{F}}$ where V is a finite set of nodes ($n,n_1,\ldots$ range over nodes); P is a finite set of ports ($p,p_1,\ldots$ range over ports); 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 $\mathscr{F}$ is a set of functions $Connect\colon E \to P \times P$, $Attach\colon P \to V$ and $\mathscr{L}\colon V \cup P \cup E \to D$ such that*

- *for each edge $e \in E$, $Connect(e)$ is the pair $\{p_1,p_2\}$ of ports connected by e (an ordered pair if the edge is oriented);*
- *for each port $p \in P$, $Attach(p)$ is the node to which p belongs.*
- *$\mathscr{L}$ is a labelling function that returns a record for each element in $V \cup P \cup E$, such that for each node $n \in V$, $\mathscr{L}(n)$ contains an attribute Interface whose value is the list of names of the ports attached to n, that is, $\mathscr{L}(n).Interface = [\mathscr{L}(p_i).Name \mid Attach(p_i) = n]$, satisfying the following constraint: $\mathscr{L}(n_1).Name = \mathscr{L}(n_2).Name \Rightarrow \mathscr{L}(n_1).Interface = \mathscr{L}(n_2).Interface$.*

Similarly, we call Interface *of a graph its set of free ports (i.e., ports that do not have edges attached to them).*

The functions *Connect* and *Attach* can be implemented as attributes in the records associated to edges and ports, respectively.

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 edges involving the arrow node indicate how the instance of R should be linked to the remaining part of the graph in a rewriting step (they specify the rule morphism in the SPO approach). 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. 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 [21] 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 (Port-graph Morphism) *Let $G = (V_G, P_G, E_G, D_G)_{\mathscr{F}_G}$ and $H = (V_H, P_H, E_H, D_H)_{\mathscr{F}_H}$ be port graphs over the same signature ∇, a* morphism *f from G to H, denoted $f : G \to H$, is a family of (partial) functions $\langle f_V : V_G \to V_H, f_P : P_G \to P_H, f_E : E_G \to E_H, f_D : D_G \to D_H \rangle$ such that*

- *f_V, f_P, f_E are injective, i.e., the morphism does not identify distinct nodes, ports or edges;*
- *$\forall e \in E_G$, if $Connect_G(e) = \{p_1, p_2\}$ then $\{f_P(p_1), f_P(p_2)\} = Connect_H(f_E(e))$, i.e., the morphism preserves the edge connections;*
- *$\forall n \in V_G$, if $Attach_G(p) = n$ for some p then $f_V(n) = Attach_H(f_P(p))$, i.e., the morphism preserves the port attachments;*
- *For all $n \in Dom(f)$, $f_D(\mathscr{L}_G(n)) = \mathscr{L}_H(f_V(n))$*
  For all $p \in Dom(f)$, $f_D(\mathscr{L}_G(p)) = \mathscr{L}_H(f_P(p))$
  For all $e \in Dom(f)$, $f_D(\mathscr{L}_G(e)) = \mathscr{L}_H(f_E(e))$,
  i.e., the morphism preserves attributes and their values; note that f_D may instantiate variables.

We denote by $f(G)$ the subgraph of H consisting of the set of nodes, ports, edges and records that are images of nodes, ports, edges and records in G.

This definition ensures that each corresponding pair of nodes, ports and edges in G and H have the same set of attribute labels and associated values, except at positions where there are variables. When using this definition to define rewriting, the left-hand side of the rewrite rule may include variable labels but the graph to be rewritten will not have variables.

Definition 5 (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 $G - g(L)$.*

Graph rewriting systems can be given a categorical semantics: the most popular approaches are the single pushout and double pushout semantics [18]. In this paper we are interested in the category C whose objects are attributed hierarchical port graphs and whose morphisms are hierarchical morphisms (defined in the next section). We recall now the notion of a pushout from [16], which is used to define rewriting steps.

Definition 6 (Pushout) *A pushout in category C is a tuple (a_1, a_2, b_1, b_2) of morphisms $a_i : X \to X_i$, $b_i : X_i \to X'$ such that $b_1 \circ a_1 = b_2 \circ a_2$ and for all $b'_i : X_i \to C$ where $i = \{1|2\}$ such that $b'_2 \circ a_2 = b'_1 \circ a_1$ there exists a unique morphism $c : X' \to C$ that satisfies $c \circ b_1 = b'_1$ and $c \circ b_2 = b'_2$.*

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 5 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 $g(R)$.

Under suitable conditions (see [21] for details), the rewriting relation can be given a semantics using the SPO construction: a rewriting step is obtained by computing the pushout of the rule morphism (defined by the arrow node and its edges) and the matching morphism. $\mathscr{S}$ can be applied to x we say that the strategy has failed.

PORGY [21] is an interactive environment that includes functionality to create port graphs and port graph rewrite rules, and to apply rules to graphs (according to user-defined strategies). In this implementation, the functions *Connect* and *Attach* (see Definition 3) are represented as attributes in records (i.e., records contain data attributes, visualisation attributes such as colour or shape, and structural attributes such as Connect and Attach). PORGY also provides a visual representation of the rewriting derivations, 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 function on nodes, called *Ladder*, which returns a graph. This function will be implemented as a new kind of attribute in records associated with nodes, which we also call *Ladder* and 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).

The set of attributed hierarchical port graphs will be defined by induction. 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 graphs defines a tree structure. 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 by other graphs and the resulting overall graph remains well formed; it is more difficult to ensure that no edges are left dangling if there are edges across components), and facilitates the visualisation of the graph, generating a greater ease in user-understanding. More flexible notions of hierarchical graphs will be considered in future work.

Definition 7 (AHP-Signature) *An AHP-signature $\nabla_{\mathscr{G}}$ consists of a port graph signature (see Definition 1), where the set of variables includes elements of type graph. The subset $\mathscr{X}_{\mathscr{V}}^{G}$ of $\mathscr{X}_{\mathscr{V}}$ consists of variables of type graph, denoted $\mathfrak{X}_1, \mathfrak{X}_2, \ldots$, representing unknown port graphs, each with an associated interface (list of port names) denoted by Interface$(\mathfrak{X})$.*

Definition 8 (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 or a variable $\mathfrak{X}$ of type graph.

An AHP at level $i+1$ is a tuple $(V, P, E, \mathscr{G}, D)_{\mathscr{F}}$ consisting of a set V of nodes, a set P of ports, a set E of edges, a set $\mathscr{G} \subseteq \bigcup_{j \leq i} \mathscr{H}_j$ of AHP at a lower level, and a set D of records over $\nabla_{\mathscr{G}}$, together with a set of functions Connect, Attach, Ladder and $\mathscr{L}$ such that Connect and Attach are defined as in Definition 3, Ladder$\colon V \mapsto \mathscr{G}$ is a partial injective function mapping nodes to lower-level graphs that must have the same interface as the node (i.e., the Ladder graph should have same number of free ports, with the same names and attributes, as the node), and $\mathscr{L}$ is a labelling function that associates records

in D to elements in $V, P, E, \mathscr{G}$. Given an AHP G, we assume that the Ladder graphs within G are pairwise disjoint and also disjoint with the top level graph (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 whose value is the graph shown in the centre of the figure, which in turn contains a node *Pools* with a Ladder graph (the graph in the rightmost part of the figure).

To define the rewriting relation, we need AHP-morphisms. The definition of AHP-morphism (Definition 9) 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, ladder graphs and records that are images of nodes, ports, edges, ladders and records in G.

Definition 9 (AHP Morphism) *AHP-morphisms are defined inductively.*

An AHP at level 0 is either an attributed port graph or a variable of type graph, consequently an AHP-morphism at level 0 is a port graph morphism or a mapping from graph variables to AHP.

Let $G = (V_G, P_G, E_G, \mathscr{G}_G, D_G)_{\mathscr{F}_G}$ and $H = (V_H, P_H, E_H, \mathscr{G}_H, D_H)_{\mathscr{F}_H}$ be two AHP graphs over the same signature $\nabla_{\mathscr{G}}$; assume w.l.g. that G is at level i. A (partial) AHP morphism at level *i, f, from G to H, denoted $f : G \to H$, with definition domain $Dom(f)$, is defined by a family of (partial) functions $\langle f_V : V_G \to V_H, f_P : P_G \to P_H, f_E : E_G \to E_H, f_{\mathscr{G}} : \mathscr{G}_G \mapsto \mathscr{G}_H, f_D : D_G \mapsto D_H \rangle$ such that*

- *$f_V, f_P, f_E, f_{\mathscr{G}}$ are injective.*
- *For all $e \in E$, if $Connect_G(e) = \{p_1, p_2\}$ then $Connect_H(f_E(e)) = \{f_P(p_1), f_P(p_2)\}$ (i.e., the morphism preserves the edge connections).*
- *For all $n \in V_G$, if $Attach_G(p) = n$ for some p then $Attach_H(f_P(p)) = f_V(n)$ (i.e., the morphism preserves the attachment of ports to nodes), and if $Ladder_G(n) = W$ then $Ladder_H(f_V(n)) = f_{\mathscr{G}}(W)$, where $f_{\mathscr{G}}$ is an AHP morphism at level $j < i$ (i.e., the morphism preserves the hierarchical structure).*
- *For all $n \in Dom(f)$, $f_D(\mathscr{L}_G(n)) = \mathscr{L}_H(f_V(n))$*
  For all $p \in Dom(f)$, $f_D(\mathscr{L}_G(p)) = \mathscr{L}_H(f_P(p))$
  For all $e \in Dom(f)$, $f_D(\mathscr{L}_G(e)) = \mathscr{L}_H(f_E(e))$
  For all $W \in Dom(f)$, $f_D(\mathscr{L}_G(W)) = \mathscr{L}_H(f_{\mathscr{G}}(W))$
  This constraint ensures that the morphism preserves record attributes and their values; note that f_D may instantiate variables.

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 10 (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. 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 11 (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 a total 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 between L and G, if a node in L has a ladder graph W then a new morphism is sought (recursively, following the inductive definition of AHP morphism) between W and the ladder graph in the corresponding node in G.

The *rewriting steps generated by AHP rules*, denoted $G \Rightarrow H$, are 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. We study properties of the rewriting relation and provide a SPO semantics in Section 5, after giving examples of applications in the next section.

4 Applications

Securitisation. As defined in [33] "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. The ABS trading model described in [3] has been specified as a port graph rewriting system and implemented in PORGY [19]; in this paper we incorporate lower, more operational tiers into the model, by means of AHP. The operational tiers will lead to the computation of the asset'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. AHP 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 [19] sits at the top level of the model hierarchy and below this system lie several more deterministic subsystems that encapsulate asset origination, packaging, structuring and servicing processes, and therefore aid in enforcing internal checks as a result of complete system integration. Figure 1 depicts the main components of the securitisation model. The first, second and third graphs respectively represent (simplified versions of) the Secondary Market, Structuring and Origination tiers respectively. B is an agent, the buyer or seller bank (there are ten nodes representing banks in the top level), node A is an asset-backed security, node Tranches represents the internal structuring or content of an asset and Pools the connection to underlying that provide an income stream.

The hierarchical rewrite rules that drive execution have patterns that cut across the two or three identified tiers. The transformation of the context graph can be seen in asset transfers, general node movements, and colour changes that indicate changes of state. Rules such as that which aid in the structuring of the asset as seen in Figure 2 or its flattened version as seen in Figure 3, can be applied from within the system.

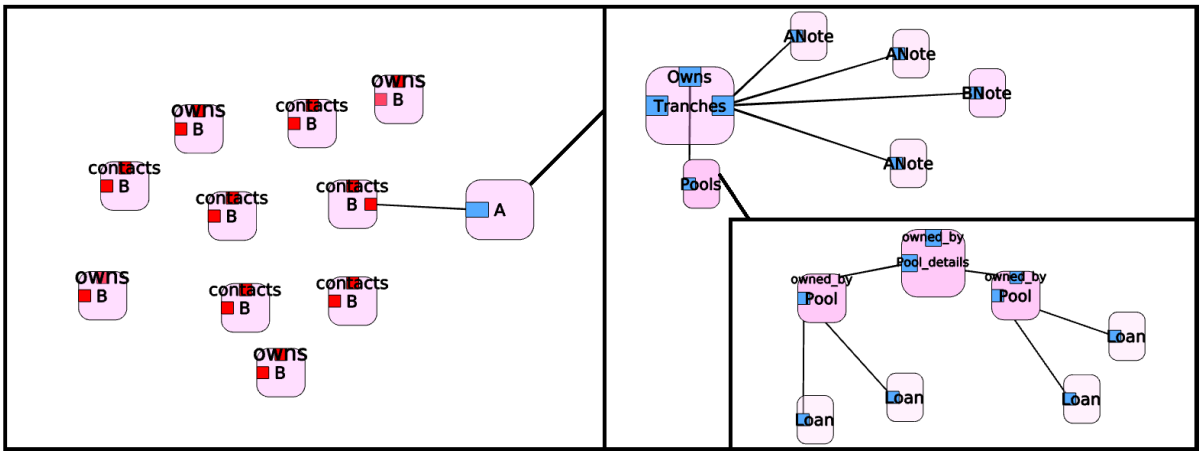

Figure 1: Sample Starting Graph

A flattening algorithm that transforms AHP into standard attributed port graphs 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.

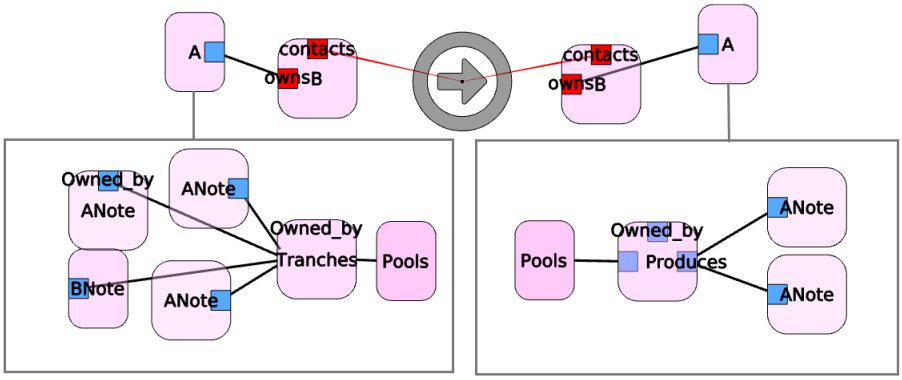

Figure 2: Sample Hierarchical Rewrite Rule: “Update Ladder”

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

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

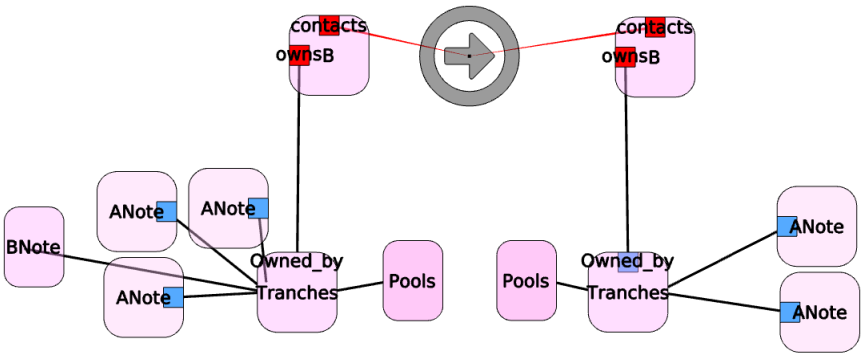

Figure 3: Flattened Version of Sample Hierarchical Rewrite Rule: "Update Ladder". The Ladder graph of node A is updated.

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. To palliate this problem, several extensions of interaction nets have been proposed (see e.g., [1, 2, 22]). In [2], a representation of intuitionistic logic proofs (or λ-terms) is given by means of *higher-order port-graphs* (HOPG).

HOPG [23] 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.

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 nesting structure that can be recursively flattened on demand to produce a conventional graph. Similar to the example HOPG-implementation given in [23], an AHP graph can directly represent a proof (or a λ-term) that contains a box by using a hierarchical node *Box* whose Ladder contains the box structure.

In general, to encode an HOPG as an AHP graph we simulate higher-order variable nodes 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 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 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.

The encoding of HOPG rules is more involved; a detailed definition is left for future work.

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

Let $G = (V_G, P_G, E_G, \mathscr{G}_G, D_G)_{\mathscr{F}_G}$ be an AHP graph without variables of type graph,
let V_G^0 be the subset of V_G where the function Ladder is not defined and P_G^0 the set of ports attached to nodes in V_G^0.
$\mathscr{T}(G) = G$ if $V_G^0 = V_G$ (i.e., $G \in \mathscr{H}_0$),
$\mathscr{T}(G) = G'$ otherwise, where $G' = (V_{G'}, P_{G'}, E_{G'}, D_{G'})_{\mathscr{F}_{G'}}$ such that
$\mathscr{W} = \{\mathscr{T}(Ladder(n)) \mid n \in dom(Ladder_G)\}$
$V_{G'} = V_G^0 \cup \bigcup_{W \in \mathscr{W}} V_W$
$P_{G'} = P_G^0 \cup \bigcup_{W \in \mathscr{W}} P_W$
$E_{G'} = E_G \cup \bigcup_{W \in \mathscr{W}} E_W$
$D_{G'} = D_G \cup \bigcup_{W \in \mathscr{W}} D_W$
$Connect_{G'} = \overline{Connect}_G \cup \bigcup_{W \in \mathscr{W}} Connect_W$
$Attach_{G'} = Attach_G|_{P_{G'}} \cup \bigcup_{W \in \mathscr{W}} Attach_W$
$\mathscr{L}_{G'} = \mathscr{L}_G|_{V_{G'} \cup P_{G'} \cup E_{G'}} \cup \bigcup_{W \in \mathscr{W}} \mathscr{L}_W$

5 Properties

Soundness: Rewriting an AHP produces another AHP, that is, rewriting does not leave dangling edges and maintains a hierarchical structure.

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.

Proof. We need to check that there are no dangling edges after rewriting, and the result is a hierarchical structure.

The restriction to *injective* Ladder functions in the definition of AHP ensures that the matching morphism will not identify two ladder graphs (which would break the hierarchy). Moreover, 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 therefore the replacement of the subgraph $g(L)$ by $g(R)$ maintains a hierarchical structure: the rewiring (during rewriting) cannot introduce edges across subgraphs at different level.

A rewriting step cannot leave dangling edges due to the constraints imposed in the definition of rewriting, similar to the constraints used in the definition of regular port graph rewriting: the rewiring phase takes care of the edges arriving from outside the redex to images of ports connected to the arrow node, and there are no other edges between ports outside the redex and ports in the redex (as specified in the definition of rewriting). Hence, there are no dangling edges when $g(L)$ is replaced with $g(R)$ in an AHP-rewriting step. □

Flattening: 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 AHP without graph-type variables into a regular port graph as seen in Algorithm 1. Since the Ladder graph of a node n has the *same interface* as n, we can flatten the graph G 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 graph: the function $\overline{Connect}_G$ in Algorithm 1 is similar to $Connect_G$ except that if $e \in E_G$ was connected to a port p in a node

n with a ladder graph W, then $\overline{Connect}_G(e)$ returns the port corresponding to p in the flattened version of W (instead of p), which exists since the ladder graph W has the same interface as n. In other words, $\mathscr{T}(G) = G'$ where G' is obtained from G by replacing each hierarchical node n in G with $\mathscr{T}(Ladder_G(n))$ and connecting edges incident to ports in n to the corresponding ports in $\mathscr{T}(Ladder_G(n))$.

The recursive definition of the flattening function given in in Algorithm 1 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 attributed 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 $\mathscr{T}(L \Rightarrow_C R)$.*

Proof.

1. The first part follows by induction: The base case is trivial: G is an AHP at level 0, that is, a regular port graph (since there are no variables of type graph by assumption), and the translation returns the same graph.

   If G is an AHP at level i, then by induction (since the ladder graphs in G are at a lower level) the set $\mathscr{W}$ computed by the flattening function contains regular attributed port graphs. Moreover, they are disjoint because by definition of AHP all the ladder graphs are disjoint. The output of the flattening function is built using the port graphs in $\mathscr{W}$ and the nodes in G that do not contain ladders. To complete the proof we need to show that the functions $Connect_{G'}$, $Attach_{G'}$ and $\mathscr{L}_{G'}$ are well defined.

   We prove first that the function $Connect_{G'}$ returns a pair of ports in G' for each edge in $E_{G'}$:

   All the edges in G' are either in E_G or in a graph $W \in \mathscr{W}$ (i.e. in the translation of a ladder graph). If $e \in E_G$, by definition $\overline{Connect}_G$ is defined and returns the ports where the edge is attached (if e was connected in G to a port p in a node n with a ladder graph, then $\overline{Connect}_G$ returns the port in W corresponding to p, which exists since the ladder graph has the same interface as n). If $e \in W$ then $Connect_{G'}(e) = Connect_W(e)$ by definition of $\mathscr{T}$, and $Connect_W$ is well defined by induction.

   To show that $Attach_{G'}$ returns a node in $V_{G'}$ for each port p in P'_G, we observe that the ports in $P_{G'}$ are either in P^0_G and therefore $Attach_G(p)$ is defined, or in one of the graphs W obtained by flattening a ladder graph, in which case $Attach_W(p)$ is defined by induction.

   Finally, it is easy to see that $\mathscr{L}_G$ is well defined by induction and the assumption that G is an AHP.

   Since AHP rules are AHP port graphs, their flattened version is a regular port graph. To show that it is indeed a port graph rewrite rule it is sufficient to notice that the arrow node does not have a Ladder graph, and therefore it is part of the flattened rule, together with the edges that link it to the left and right hand sides, as expected.

2. The second part is a consequence of the fact that the same flattening function is applied to the rule and to the graph that will be rewritten; if there is an AHP matching morphism between the AHP L and G, then there is a matching morphism between $\mathscr{T}(L)$ and $\mathscr{T}(G)$.

□

SPO rewriting semantics: Löwe [30] showed that if *Sig* is an algebraic signature, all *Sig*-algebras and partial *Sig*-morphisms form a category $Alg^p(Sig)$. This category is not closed with respect to pushouts in general. However, if *Sig* contains only unary operators, i.e., it is a *graph structure* in Löwe's terminology, then pushouts do exist (see [30], Theorem 2.7). This result also holds for attributed graph structures as shown in [31]. Attributed port graphs have been shown to be attributed graph structures in [21], by defining suitable signatures and interpreting an attributed port graph as an algebra with that signature. Here we show that AHPs are also attributed graph structures. The proof is more involved for AHPs due to the nesting of graphs. The key idea is to introduce a sort "graph" and operators to group nodes, and hence ports and edges into graphs at various levels.

First, we recall the definition of algebraic signature and graph structure and refer to [30,31] for more details.

Definition 12 (Graph Structure) *An algebraic signature $Sig = (S, Op)$ consists of a set S of* sorts *and a set Op of operator symbols.*

Given an algebraic signature $Sig = (S, Op)$, if A, B are Sig-algebras, a partial Sig-morphism $h : A \mapsto B$ is a total morphism from some sub-algebra A_h of A to B; A_h is called the scope of h.

A graph structure *is a signature that contains unary operators only.*

*If $GS = (S_1, OP_1)$ is a graph structure, S a subset of S_1 and $SIG = (S_2, OP_2)$ an arbitrary signature, a SIG-*attribution *of GS is an S-indexed family of operator symbols $ATTROP = (ATTROP_s : s \mapsto s2_s)_{s \in S}$ where $s \in S$ and $s2 \in S_2$.*

An attributed graph *is a GS-graph with attributes in SIG, i.e., an algebra with respect to the signature $ATTR = GS + SIG + ATTROP$.*

A morphism $f : A \mapsto B$ between GS-graphs A and B having attributes in SIG is a partial GS-morphism $f1 : (A)_{GS} \mapsto (B)_{GS}$ together with a total SIG-morphism $f2 : (A)_{SIG} \mapsto (B)_{SIG}$ satisfying for all operators $attr : s1 \mapsto s2 \in ATTROP$ and all $x \in A(f1)_{s1}, f2(attr^A(x)) = attr^B(f1(x))$.

A rewrite rule is an ATTR-morphism r whose SIG-component is an isomorphism $f2 : (A)_{SIG} \mapsto (B)_{SIG}$.

To show that AHP are attributed graph structures, we consider signatures $AH = (S, OP)$ and $SIG = (S_1, OP_1)$ such that:

- $S = \{node, port, edge, graph, rec_{node}, rec_{port}, rec_{edge}, rec_{graph}, list[port]\}$. Formally, $list[port]$ is a family of sorts, one for each arity; we abbreviate it as one sort.
- OP is the set of operators including:
  1. $s, t : edge \mapsto port$; these operators will be interpreted by the *Connect* function.
  2. $ports : node \mapsto list[port]$; this operator will be interpreted by using the *Attach* function (again, formally it is a family of operators, one for each arity, but we abbreviate it as one operator).
  3. $ladder_V : node \mapsto graph$; this operator will be interpreted as a function that returns the ladder graph to which the node belongs.
  4. $l_V : rec_{node} \mapsto node$, $l_P : rec_{port} \mapsto port$, $l_E : rec_{edge} \mapsto edge$, $l_{\mathcal{G}} : rec_{graph} \mapsto graph$; these operators will be interpreted as functions that return the node, port, edge or graph to which the record belongs.
- $S_1 = D \cup \{attribute, value, pair, record\}$, where D is the set of data sorts (for *values*), the other sorts are used to type the operators that build records.

- OP_1 includes a set of operators on data sorts (such as arithmetic operators) and operators used to build records:
  1. $_ := _ : attribute, value \mapsto pair$
  2. $_._ : record, attribute \mapsto value$
  3. $\{_,_\} : pair, record \mapsto record$
  4. $\{\} : \mapsto record$

Let the *SIG*-attribution of *AH*, $ATTROP$, be such that $ATTROP_{rec_{node}} : rec_{node} \mapsto record$, $ATTROP_{rec_{port}} : rec_{port} \mapsto record$, and similarly for the other *rec* sorts. We show below that an AHP G can be seen as an algebra on the combination of three signatures $ATTR = AH + SIG + ATTROP$ and is therefore an attributed graph structure.

Property 3 *AHP graphs are attributed graph structures.*

Proof. Since all operators in *OP* are unary, *AH* is a graph structure. To complete the proof we show that an AHP $G = (V, P, E, \mathscr{G}, D)_{\mathscr{F}}$ can be seen as an algebra on the signature $ATTR = AH + SIG + ATTROP$:

- $V, P, E, \mathscr{G}$ are carriers of the sorts $node, port, edge, graph$ and the sort rec_i ($i \in \{node, port, edge, graph\}$) is interpreted by a set of pointers, one for each element of V, P, E, $\mathscr{G}$.
- s and t are interpreted by the function *Connect*:
  $s,t : edge \mapsto port$ such that $s(e) := p_1, t(e) := p_2$ iff $e \in E \wedge Connect(e) = (p_1, p_2)$.
- $ports$ is interpreted by a function such that if $Attach(p_i) = n$ $(1 \leq i \leq k)$ then $ports(n) := [p_1, \ldots, p_k]$.
- $ladder_V$ is interpreted by a function such that if n is a node in a ladder graph W $(n \in V_W)$ then $ladder_V(n) = W$.
- the injective operators l_V, l_P, l_E, $l_{\mathscr{G}}$ are interpreted using the respective AHP's labelling function $\mathscr{L}$ (for example, if $\mathscr{L}_G(e) = r$ then $l_E(r) = e$; the definitions of l_V and l_P are similar).
- The sub-algebra that corresponds to *SIG* defines the interpretation for records (it can either be interpreted as a term-algebra with variables or a class of concrete objects, depending on whether we are considering a graph in a rewrite rule or a concrete graph to be rewritten).

This completes the proof. □

We can also show that AHP morphisms are ATTR morphisms. To complete the SPO semantics for rewriting, we need to show that AHP rules define a partial morphism from the left to the right-hand side. This was shown in [21] for *simple* rules where *the arrow node maps one port in the left-hand side to only one port in the right-hand side*. Under the same condition, a similar result can be shown for AHP rules. The proof is more involved because there is an additional sort *graph* and operators that map nodes to ladder graphs. However, since the partial morphism defined by the rule is generated by the map between ports in the top level of L and R (specified by the arrow node and its edges), no ladder graphs are involved (top level ports do not belong to ladder graphs, since all graph components in an AHP are disjoint).

Property 4
1. *A simple AHP rule is an $ATTR$-morphism r whose SIG component is an isomorphism, and a match m between the left-hand side of a rule, L, and a redex in an AHP G is an $ATTR$ morphism whose AH component is total.*
2. *The application of a simple rule r to a graph G at redex $m(L)$ is the pushout of r and m.*

Proof. To prove the first part, we first show that an AHP morphism between two AHP graphs (Definition 9) maps to a partial morphism between AH-graphs A and B that have attributes in SIG, together with a total SIG-morphism satisfying for all operators $attr$: $s1 \mapsto s2 \in ATTROP$ and all $x \in A(f1)_{s1}, f2(attr^A(x)) = attr^B(f1(x))$:
In the AHP-morphism definition, a (partial) *morphism* f from G to H, denoted $f : G \to H$, with definition domain $Dom(f)$, is defined by a family of partial functions $\langle f_V : V_G \to V_H, f_P : P_G \to P_H, f_E : E_G \to E_H, f_{\mathscr{G}} : \mathscr{G}_G \mapsto \mathscr{G}_H, f_D : D_G \mapsto D_H \rangle$. This family of functions define a partial AH-morphism $f_1 : (G)_{AH} \mapsto (H)_{AH}$ which coincides with $\langle f_V : V_G \to V_H, f_P : P_G \to P_H, f_E : E_G \to E_H, f_{\mathscr{G}} : \mathscr{G}_G \mapsto \mathscr{G}_H \rangle$ on the carriers of sorts *node, port, edge, graph*. The total SIG-morphism $f_2 : (G)_{SIG} \mapsto (H)_{SIG}$ is the restriction of f on records and coincides with $f_D : D_G \mapsto D_H$. f_2 satisfies $\forall attr : a' \mapsto a \in ATTROP$ and all $G(f_1)_s, f_2(attr^G(x)) = attr^H(f_1(x))$ due to the condition on $\mathscr{L}$ in definition 9 such that the morphism preserves record attributes and their values:
For all $n \in Dom(f)$, $f_D(\mathscr{L}_G(n)) = \mathscr{L}_H(f_V(n))$
For all $p \in Dom(f)$, $f_D(\mathscr{L}_G(p)) = \mathscr{L}_H(f_P(p))$
For all $e \in Dom(f)$, $f_D(\mathscr{L}_G(e)) = \mathscr{L}_H(f_E(e))$
For all $W \in Dom(f)$, $f_D(\mathscr{L}_G(W)) = \mathscr{L}_H(f_{\mathscr{G}}(W))$.

To show that a simple AHP rule defines a partial morphism between the left- and right-hand sides, notice that in a simple rule the arrow node links one port in L to at most one port in R. Thus, the edges in the arrow node define a *partial function* from P_L (the set of ports in the left-hand side) to P_R (the set of ports in the right-hand side). Since there are no operators on ports in the algebra, this is trivially a morphism.

Since records are implemented in the same way on both sides, an AHP rule can be seen as an ATTR-morphism whose SIG component is an isomorphism. This completes the proof of the first part.

To prove the second part, first notice that a match m between the left-hand side of a rule, L, and a redex in an AHP G is an ATTR-morphism whose AH-component is total. This follows from the definition of AHP morphism (Definition 9).

The fact that a pushout of r and m is equivalent to the application of a simple rule r to G at redex $m(L)$ in order to produce H is seen in the description of the steps involved in an AHP rewrite step $G \Rightarrow H$, which map to the steps described in [31] to build the pushout object.

The gluing object consists of the ports in L that are connected to the bridge ports. The pushout object is isomorphic to G where $m(L)$ is replaced with $m(R)$ and the external edges connected to $m(dom(r))$ are redirected as indicated in the definition of rewriting step. □

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, 36]) 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 [37]. 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 [24] to represent proofs, and gs-graphs [10] can be proven to be essentially equivalent.

Hierarchical hyper-graphs [16] also provide structuring features: attributes of type graph are permit-

ted 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. A less restrictive version can be found in [39].

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 [11] model hierarchy by labelling nodes of set N, over a set of atoms A, or permitting an embedded directed sub-graph created from N within A. 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 [11] 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. A similar framework can be found in [12].

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 [42]. Other interesting hierarchical representations include that of L-Graphs in which edges and nodes are labelled by graphs [41], $\mathscr{M}$-adhesive Graph Transformation Systems [38] that provide a generic algebraic definition to cater for varying graphs types, that of the verbose property graphs used in graph analytics [28], and the implementation found in [45]; that also makes use of attributes and multi-typed sub-graphs. [38] also contains details of multi-hierarchical graphs and graph groupings.

Specifically for port graphs, an abstract higher-order calculus inspired by the ρ-calculus [13] is defined in [6, 7], 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 matching and rewriting) is first-order. As already mentioned, the higher-order port graphs found in [23] 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, modular and feasible. Future work will include a full implementation of this design within PORGY, completing the construction of the case-studies outlined and developing a translation between HOPG and AHP formalisms. Our hierarchical constraint could be relaxed to permit edges linking nodes in graphs at different levels in the hierarchy, although this will make the implementation of matching more involved. Boundary-crossing edges are permitted within some of the visual languages used in software modelling (e.g., to represent UML diagrams).

References

[1] Beniamino Accattoli (2015): *Proof nets and the call-by-value λ-calculus*. Theor. Comput. Sci. 606, pp. 2–24, doi:10.1016/j.tcs.2015.08.006.

[2] Sandra Alves, Maribel Fernández & Ian Mackie (2011): *A new graphical calculus of proofs*. In: Proceedings 6th International Workshop on Computing with Terms and Graphs, TERMGRAPH 2011, Saarbrücken, Germany, 2nd April 2011., pp. 69–84, doi:10.4204/EPTCS.48.8.

[3] Kartik Anand, Alan Kirman & Matteo Marsili (2013): *Epidemics of rules, rational negligence and market crashes*. The European Journal of Finance 19(5), pp. 438–447, doi:10.1080/1351847X.2011.601872.

[4] Oana Andrei (2008): *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)*. Ph.D. thesis, National Polytechnic Institute of Lorraine, Nancy, France.

[5] Oana Andrei, Maribel Fernández, Hélène Kirchner, Guy Melançon, Olivier Namet & Bruno Pinaud (2011): *PORGY: Strategy-Driven Interactive Transformation of Graphs*. In: TERMGRAPH, pp. 54–68, doi:10.4204/EPTCS.48.7.

[6] Oana Andrei & Hélène Kirchner (2008): *A Rewriting Calculus for Multigraphs with Ports*. Electr. Notes Theor. Comput. Sci. 219, pp. 67–82, doi:10.1016/j.entcs.2008.10.035.

[7] Oana Andrei & Helene Kirchner (2009): *A Higher-Order Graph Calculus for Autonomic Computing*. In Marina Lipshteyn, Vadim E. Levit & Ross M. McConnell, editors: Graph Theory, Computational Intelligence and Thought, Springer-Verlag, Berlin, Heidelberg, pp. 15–26, doi:10.1007/978-3-642-02029-2_2.

[8] Hendrik Pieter Barendregt (1990): *Functional Programming and Lambda Calculus*. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B), pp. 321–363, doi:10.1016/B978-0-444-88074-1.50012-3.

[9] Lars Birkedal, Troels Christoffer Damgaard, Arne J. Glenstrup & Robin Milner (2007): *Matching of Bigraphs*. Electr. Notes Theor. Comput. Sci. 175(4), pp. 3–19, doi:10.1016/j.entcs.2007.04.013.

[10] Roberto Bruni, Ugo Montanari, Gordon D. Plotkin & Daniele Terreni (2014): *On Hierarchical Graphs: Reconciling Bigraphs, Gs-monoidal Theories and Gs-graphs*. Fundam. Inform. 134(3-4), pp. 287–317, doi:10.3233/FI-2014-1103.

[11] Giorgio Busatto & Berthold Hoffmann (2001): *Comparing Notions of Hierarchical Graph Transformation*. Electr. Notes Theor. Comput. Sci. 50(3), pp. 310–317, doi:10.1016/S1571-0661(04)00184-7.

[12] Giorgio Busatto, Hans-Jörg Kreowski & Sabine Kuske (2005): *Abstract hierarchical graph transformation*. Mathematical Structures in Computer Science 15(4), pp. 773–819, doi:10.1017/S0960129505004846.

[13] Horatiu Cirstea & Claude Kirchner (2001): *The rewriting calculus — Part I* and *II*. Logic Journal of the Interest Group in Pure and Applied Logics 9(3), pp. 427–498, doi:10.1093/jigpal/9.3.339.

[14] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel & Michael Löwe (1997): *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, World Scientific, pp. 163–246, doi:10.1142/9789812384720_0003. Available at `http://www.worldscientific.com/doi/abs/10.1142/9789812384720_0003`.

[15] Vincent Danos, Feret, Walter Fontana, Russell Harmer, Jonathan Hayman, Jean Krivine, Chris Thompson-walsh & Glynn Winskel: *Graphs, rewriting and causality in rule-based models*, doi:http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.221.6822.

[16] Frank Drewes, Berthold Hoffmann & Detlef Plump (2002): *Hierarchical Graph Transformation*. Journal of Computer and System Sciences 64(2), pp. 249 – 283, doi:10.1006/jcss.2001.1790.

[17] Niels Van Eetvelde & Dirk Janssens (2003): *A Hierarchical Program Representation for Refactoring*. Electr. Notes Theor. Comput. Sci. 82(7), pp. 91–104, doi:10.1016/S1571-0661(04)80749-7.

[18] Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner & Andrea Corradini (1997): *Algebraic Approaches to Graph Transformation - Part II: Single Pushout Approach and Comparison with Double Pushout Approach*. In: *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pp. 247–312, doi:10.1142/9789812384720_0004.

[19] Nneka Ene, Maribel Fernández & Bruno Pinaud: *Graph Models for Capital Markets*, doi:https://nms.kcl.ac.uk/nneka.ene/papers.html. Available from https://nms.kcl.ac.uk/nneka.ene/papers.html.

[20] Gregor Engels & Reiko Heckel (2000): *Graph Transformation as a Conceptual and Formal Framework for System Modeling and Model Evolution*, pp. 127–150. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/3-540-45022-X_12.

[21] Maribel Fernández, Hélène Kirchner & Bruno Pinaud (2017): *Strategic Port Graph Rewriting: an Interactive Modelling Framework*. Research Report, Inria ; LaBRI - Laboratoire Bordelais de Recherche en Informatique ; King's College London, doi:https://hal.inria.fr/hal-01251871. Available at `https://hal.inria.fr/hal-01251871`.

[22] Maribel Fernández, Ian Mackie & Jorge Sousa Pinto (2007): *A Higher-Order Calculus for Graph Transformation*. *Electr. Notes Theor. Comput. Sci.* 72(1), pp. 45–58, doi:10.1016/j.entcs.2002.09.005.

[23] Maribel Fernández & Sébastien Maulat (2012): *Higher-order port-graph rewriting*. In: *Proceedings 2nd International Workshop on Linearity, LINEARITY 2012, Tallinn, Estonia, 1 April 2012.*, pp. 25–37, doi:10.4204/EPTCS.101.3.

[24] Herman Geuvers & Iris Loeb (2007): *Natural deduction via graphs: formal definition and computation rules*. *Mathematical Structures in Computer Science* 17(3), pp. 485–526, doi:10.1017/S0960129507006123.

[25] Georges Gonthier, Martín Abadi & Jean-Jacques Lévy (1992): *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*, pp. 15–26, doi:10.1145/143165.143172.

[26] Gary Gorton & Andrew Metrick (2012): *Securitization*. Working Paper 18611, National Bureau of Economic Research, doi:http://www.nber.org/papers/w18611. Available at `http://www.nber.org/papers/w18611`.

[27] Annegret Habel, Jürgen Müller & Detlef Plump (2001): *Double-pushout graph transformation revisited*. *Mathematical Structures in Computer Science* 11(5), pp. 637–688, doi:10.1017/S0960129501003425.

[28] Martin Junghanns, André Petermann & Erhard Rahm (2017): *Distributed Grouping of Property Graphs with Gradoop*. In: *Proc. Datenbanksysteme für Business, Technologie und Web (BTW)*, pp. 103–122.

[29] Yves Lafont (1990): *Interaction Nets*. In: *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, January 1990*, pp. 95–108, doi:10.1145/96709.96718.

[30] Michael Löwe (1993): *Algebraic approach to single-pushout graph transformation*. *Theoretical Computer Science* 109, pp. 181–224, doi:10.1016/0304-3975(93)90068-5.

[31] Michael Löwe, Martin Korff & Annika Wagner (1993): *An Algebraic Framework for the Transformation of Attributed Graphs*. In M. R. Sleep, M. J. Plasmeijer & M. C. J. D. van Eekelen, editors: *Term Graph Rewriting*, John Wiley and Sons Ltd., Chichester, UK, pp. 185–199.

[32] Sheri Markose (2013): *Systemic risk analytics: A data-driven multi-agent financial network (MAFN) approach*. *Journal of Banking Regulation* 14(3-4), pp. 285–305, doi:10.1057/jbr.2013.10.

[33] Sheri Markose, Yang Dong & Bewaji Oluwasegun (2008): *An Multi-Agent Model of RMBS, Credit Risk Transfer in Banks and Financial Stability: Implications of the Subprime Crisis*, doi:http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.509.3202&rep=rep1&type=pdf.

[34] Sheri M. Markose, Bewaji Oluwasegun & Simone Giansante (2014): *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*, IGI Global, Hershey, U. S. A., pp. 561–590, doi:10.4018/978-1-4666-6268-1.ch030.

[35] O. Mason & M. Verwoerd: *Graph Theory and Networks in Biology*. *IET Systems Biology* 1(2), pp. 89–119, doi:10.1049/iet-syb:20060038.

[36] Robin Milner (2001): *Bigraphical reactive systems: basic theory*. Technical Report UCAM-CL-TR-523, University of Cambridge, Computer Laboratory.

[37] Robin Milner (2006): *Pure bigraphs: Structure and dynamics*. *Inf. Comput.* 204(1), pp. 60–122, doi:10.1016/j.ic.2005.07.003.

[38] Julia Padberg (2017): *Hierarchical Graph Transformation Revisited - Transformations of Coalgebraic Graphs*. In: *Graph Transformation - 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany, July 18-19, 2017, Proceedings*, pp. 20–35, doi:10.1007/978-3-319-61470-0_2.

[39] Wojciech Palacz (2004): *Algebraic hierarchical graph transformation*. *J. Comput. Syst. Sci.* 68(3), pp. 497–520, doi:10.1016/S0022-0000(03)00064-3.

[40] Detlef Plump (1998): *Term Graph Rewriting*. In Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski & Grzegorz Rozenberg, editors: *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2: Applications, Languages, and Tools*, World Scientific, pp. 3–61, doi:10.1142/9789812815149_0001. Available at `http://www.worldscientific.com/doi/abs/10.1142/9789812815149_0001`.

[41] H.J. Schneider (1993): *On categorical graph grammars integrating structural transformations and operations on labels*. *Theoretical Computer Science* 109(1), pp. 257 – 274, doi:10.1016/0304-3975(93)90070-A.

[42] Arash Shaban–Nejad & Volker Haarslev (2015): *Managing changes in distributed biomedical ontologies using hierarchical distributed graph transformation*. *International Journal of Data Mining and Bioinformatics* 11(1), pp. 53–83, doi:10.1504/IJDMB.2015.066334.

[43] Adam M. Smith, Wen Xu, Yao Sun, James R. Faeder & G.Elisabeta Marai (2012): *RuleBender: integrated modeling, simulation and visualization for rule-based intracellular biochemistry*. *BMC Bioinformatics* 13(8):S3, doi:10.1186/1471-2105-13-S8-S3.

[44] Jason Vallet, Hélène Kirchner, Bruno Pinaud & Guy Melançon (2015): *A Visual Analytics Approach to Compare Propagation Models in Social Networks*. In: *Proc. Graphs as Models, GaM 2015*, pp. 65–79, doi:10.4204/EPTCS.181.5.

[45] Grażyna Ślusarczyk, Andrzej Łachwa, Wojciech Palacz, Barbara Strug, Anna Paszyńska & Ewa Grabska (2017): *An extended hierarchical graph-based building model for design and engineering problems*. *Automation in Construction* 74, pp. 95 – 102, doi:10.1016/j.autcon.2016.11.008.