Nested Term Graphs

(Work In Progress)

Clemens Grabmayer
Department of Computer Science
VU University Amsterdam
The Netherlands
C.A.Grabmayer@vu.nl

Vincent van Oostrom
Philosophy
Utrecht University
The Netherlands
V.vanOostrom@uu.nl

We report on work in progress on ‘nested term graphs’ for formalizing higher-order terms (e.g. finite or infinite λ-terms), including those expressing recursion (e.g. terms in the λ-calculus with letrec). The idea is to represent the nested scope structure of a higher-order term by a nested structure of term graphs. Based on a signature that is partitioned into atomic and nested function symbols, we define nested term graphs both in a functional representation, as tree-like recursive graph specifications that associate nested symbols with usual term graphs, and in a structural representation, as enriched term graph structures. These definitions induce corresponding notions of bisimulation between nested term graphs. Our main result states that nested term graphs can be implemented faithfully by first-order term graphs.

1 Introduction

As an instance of the general question of how to faithfully represent structures enriched with a notion of scope using the same structures without it, we study the question how to faithfully represent higher-order term graphs using first-order term graphs.

To set the stage, we first informally recapitulate how to faithfully represent first-order terms using strings, and how to faithfully represent higher-order terms using first-order terms. The guiding intuition is that the notion of scope corresponds to a notion of context-freeness.

First-order terms can be represented using recursive string specifications (context-free grammars) such as $\{S ::= T \times U \mid T ::= 2, U ::= V + W, V ::= 3, W ::= 1\}$. The string $2 \times 3 + 1$ obtained from the specification by repeated substitution for variables[1] is not a faithful representation of the first-order term though, as the nesting structure is lost; the same string is obtained from the different first-order term $\{S ::= T + U \mid T ::= V \times W, V ::= 2, W ::= 3, U ::= 1\}$. A nameless (anonymous) alternative to recursive string specifications is to introduce a box (scope) construct in the language of strings, which indeed allows to faithfully represent the first-order terms: $\boxed{\boxed{2}\times\boxed{\boxed{3}+\boxed{1}}}$ vs $\boxed{\boxed{\boxed{2}\times\boxed{3}}+\boxed{1}}$ or after unboxing values $\boxed{2\times\boxed{3+1}}$ vs $\boxed{\boxed{2\times 3}+1}$. However, having the box construct makes this representation go beyond a string representation proper (apart from the representation quickly becoming unwieldy, on paper). A standard way to overcome this is to split the box $\square$ into[2] symbols [and] that are adjoined to the alphabet yielding the proper strings $[2 \times [3+1]]$ vs $[[2 \times 3] + 1]$. This is the common faithful representation of first-order terms as strings. Note that not just any string represents a first-order term. In particular, left and right brackets must be matching, the context-freeness aspect mentioned above, e.g. it would not do to substitute the string $3] + [1$ for X in $[2 \times X]$.

[1] Variables can be thought of as named subterms.

[2] For obvious visual reasons we use square brackets here instead of the usual parentheses. Parentheses are not needed at all when the symbols in the alphabet are enriched with arities. As shown by (Reverse) Polish Notation, arities are sufficient to capture context-freeness.

Submitted to:
TERMGRAPH 2014

Higher-order terms can be represented by using recursive first-order term specifications.[3] To illustrate this we make use of an example in functional programming (Lisp) taken from [11] concerning the unhygienic expansion of the macro $(\mathsf{or}\,\langle exp\rangle_1\langle exp\rangle_2) ::= (\mathsf{let}\, v\,[\,]_{\langle exp\rangle_1}(\mathsf{if}\, v\, v\,[\,]_{\langle exp\rangle_2}))$. Expanding this or-macro in $(\mathsf{or}\,\mathsf{nil}\, v)$ yields $(\mathsf{let}\, v\,\mathsf{nil}\,(\mathsf{if}\, v\, v\, v))$ which always yields nil due to the inadvertent capturing of v. A representation of the example by means of a recursive first-order term specification would be $\{S(v) ::= \mathsf{or}(\mathsf{nil}, v) \mid \mathsf{or}(x,y) ::= \mathsf{let}(x, T(y)), T(z) ::= \mathsf{if}(v,v,z)\}$. This representation leaves the binding effect of v in the in-part of the let implicit, by it not occurring among the arguments to T; too implicit, as repeated substitution yields $\mathsf{let}(\mathsf{nil}, \mathsf{if}(v,v,v))$. A nameless alternative to recursive first-order term specifications is to introduce a box (scope) construct in the language of first-order terms, the idea being that for every first-order term t over a vector $\vec{x}$ of n variables[4] and one additional variable, $\boxed{t}_{\vec{x}}$ is an n-ary function symbol again, e.g. $\boxed{\mathsf{if}(v,v,z)}_z$ allowing to faithfully represent the higher-order term as $\mathsf{let}(\mathsf{nil}, \boxed{\mathsf{if}(v,v,z)}_z(z))$. However, having the box construct makes this representation go beyond a first-order term representation proper. A standard way to overcome this is to split the box $\square$ into[5] unary symbols $\sqcap$ and $\sqcup$ (for opening and closing) and a nullary symbol $\bullet$ (for using the bound variable) that are adjoined to the alphabet yielding the proper first-order term $\mathsf{let}(\mathsf{nil}, \sqcap(\mathsf{if}(\bullet,\bullet,\sqcup(z))$. This is the common faithful representation of higher-order terms as first-order terms, known for the special case of λ-terms as the (extended) De Bruijn representation [3]. Note that not just any first-order term represents a higher-order term. In particular, open and close brackets must be matching, the context-freeness aspect mentioned above.

In this paper we are concerned with the same phenomenon for 'nested term graphs' in relation to their interpretations as first-order term graphs. We describe an interpretation that is faithful with regard to the respective notions of behavioral (bisimulation) semantics. As a running example we use the gletrec-expression left in Figure 2 that expresses a cyclic λ-term, and thereby a regular infinite λ-term, by means of the Combinatory Reduction System (CRS) inspired gletrec-notation. This expression corresponds to the pretty printed 'recursive graph specification' on the left in Figure 1 (the graph with scopes indicated by dotted lines). Our main result entails that the behavioral semantics of this specification is the same as that of the first-order term graph obtained from it, displayed on the right in Figure 1. Note that in this first-order term graph artefacts, additional vertices, and edges between them have been inserted to delimit scopes appropriately; they play the same rôle as the brackets in the string and term examples.

It is interesting to observe that edges connecting a bound variable to its binder seem to be forced upon us in this interpretation in order to preserve the behavorial equivalence of scopes (and their integrity; partial sharing is prevented). Interesting, as this allows for a rational reconstruction of sorts of using such edges to represent binding (instead of using variables for that purpose) as introduced in [16, 5] and common nowadays in the implementation of λ-terms.

The example in Figure 1 belongs to a particularly well-behaved subclass of recursive graph specifications that we call nested term graphs, for which the dependency between the nested symbols (n, f_1, f_2, g in the example) is tree-like. The first-order term graph is nearly a 'λ-term graph' [7], and it is closely related to a higher-order term graph [4]. For defining nested term graphs we will also consider specifications with arbitrary dependencies, allowing for both sharing and cyclicity, such as the specification left in Figure 4, which corresponds to the gletrec-expression right in Figure 2, and represents the infinite λ-term in Figure 4.

Overview. In Section 2 we define nested term graphs as such recursive term graph specifications in which

[3]In functional programming recursive first-order term specifications are known as supercombinators and the transformation of λ-terms into supercombinators is known as lambda-lifting [9].

[4]Linearity of t in $\vec{x}$ may additionally be imposed.

[5]For obvious visual reasons we use these symbols instead of the usual λ, S and 0. Decomposing a box 'vertically' into brackets here instead of 'horizontally' as before, corresponds to the matching of the brackets here being 'vertically' (along paths in the first-order term tree) whereas before it was 'horizontally' (within the string).

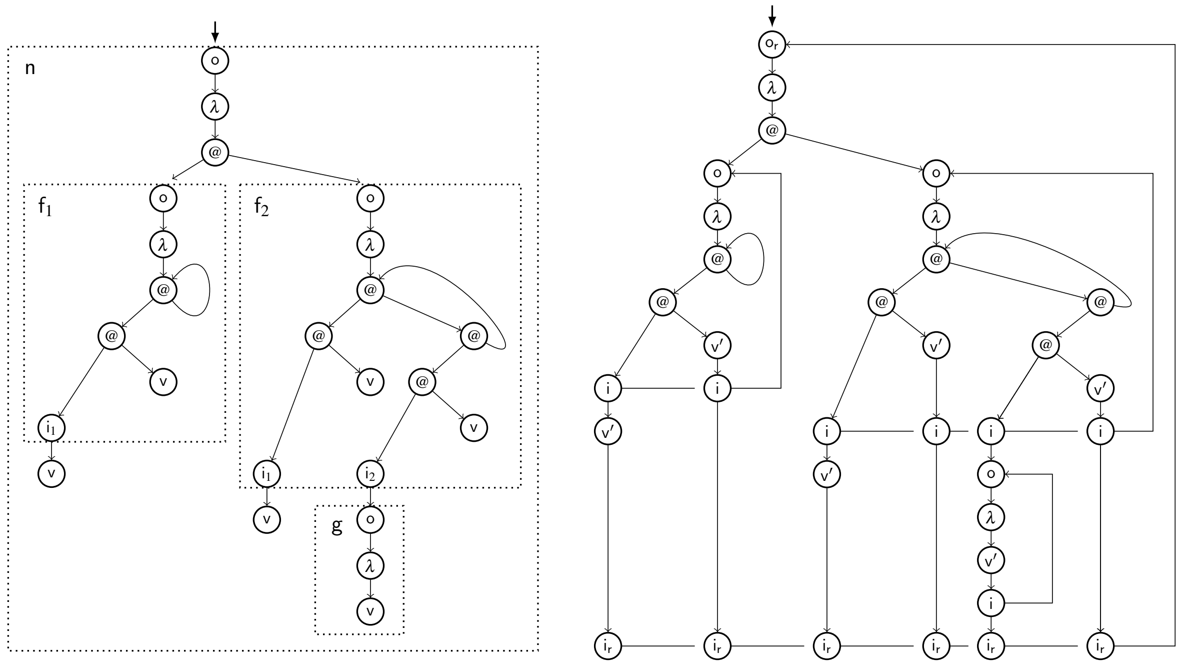


Figure 1: Pretty-printed nested term graph representing the gletrec-expression left in Figure 2, and its interpretation as a first-order term graph (back-links from i- and $\mathsf{i_r}$-labeled vertices are, typically, hinted).

$$
\begin{array}{lrcl}
\mathsf{gletrec} & \mathsf{n}() & ::= & \lambda x.\mathsf{f}_1(x)\mathsf{f}_2(x,\mathsf{g}()) \\
 & \mathsf{f}_1(X_1) & ::= & \lambda y.\mathsf{letrec}\,\alpha = X_1 y\alpha\,\mathsf{in}\,\alpha \\
 & \mathsf{f}_2(X_1,X_2) & ::= & \lambda z.\mathsf{letrec}\,\beta = X_1 z(X_2 z\beta)\,\mathsf{in}\,\beta \\
 & \mathsf{g}() & ::= & \lambda w.w \\
\mathsf{in} & \mathsf{n}() & &
\end{array}
\qquad
\begin{array}{lrcl}
\mathsf{gletrec} & \mathsf{f}() & ::= & \lambda x.\mathsf{g}(x) \\
 & \mathsf{g}(X_1) & ::= & \lambda y.\mathsf{g}(y)X_1 \\
\mathsf{in} & \mathsf{f}() & &
\end{array}
$$

Figure 2: Two CRS-inspired gletrec-expressions that represent infinite λ-terms.

the dependency 'is directly used in the definition of' between occurrences of defined (nested) symbols in the specification forms a tree. We also define structural representations of nested term graphs as integral graph structures with additional reference links, and an ancestor function that records the nesting of symbols. In Section 3 we define adequate notions of homomorphism and bisimilarity between nested term graphs in two forms: a version with a 'big-step semantics' condition for dealing with vertices labeled with defined symbols, and a 'nested' version that is based on purely local progression conditions and the use of stacks to record the nesting history. Finally in Section 4 we explain how nested term graphs can be interpreted by first-order term graphs in such a way that homomorphism and bisimilarity are preserved and reflected.

Contribution. In its present stage, our contribution is primarily a conceptual one. Inspired by Blom's higher-order term graphs [4], and by the faithful interpretation of 'λ-higher-order-term-graphs' as first-order 'λ-term-graphs' described by the first author and Rochel in [7] (which facilitates a maximal-sharing algorithm for the Lambda Calculus with letrec [8]), we set out to formalize objects with nested attributes (e.g. λ-terms with nested 'extended scopes') as enriched, and as plain, term graphs. In more detail our contribution is threefold: furnishing term graphs with a concept of nesting, developing adequate notions of behavioral semantics (homorphism, bisimulation) for nested term graphs, and describing a natural interpretation as first-order term graphs. We think that the possibility to implement higher-order features in a behavioral-semantics preserving and reflecting manner by first-order means can potentially be very fruitful.

While for the purpose of this preliminary exploration we deliberately kept to the framework of term graphs due to its simplicity, we intend to adapt the results obtained for nested term graphs also to other graph formalisms like hypergraphs, jungles, bigraphs, interaction nets, or port graphs. Also, we want to compare the concepts developed with well-known formalisms for expressing nested structures and reasoning with them, for example: bigraphs, proofnets, and Fitch-style natural-deduction proofs in predicate logic.

Preliminaries on term graphs. By $\mathbb{N}$ we denote the natural numbers including zero. For a set Σ, Σ^* stands for the set of words over alphabet Σ. We denote the empty word by ε, and write $u \cdot v$ for the concatenation of words u and v. For a word w and $i \in \mathbb{N}$, we denote by $w(i)$ its $(i+1)$-th letter, and $|w|$ for the length of w.

Let Σ be a (first-order) signature for function symbols with arity function $ar : \Sigma \to \mathbb{N}$. For a function symbol $f \in \Sigma$, we indicate by f/i that f has arity i. A *term graph over* Σ (a Σ*-term-graph*) is a tuple $\langle V, lab, args, root\rangle$ where V is a set of *vertices*, $lab : V \to \Sigma$ the *(vertex) label function*, $args : V \to V^*$ the *argument function* that maps every vertex v to the word $args(v)$ consisting of the $ar(lab(v))$ successor vertices of v (hence it holds $|args(v)| = ar(lab(v))$), and $root \in V$ is the *root* of the term graph. A term graph is called *root-connected* if every vertex is reachable from the root by a path that arises by repeatedly going from a vertex to one of its successors. By $\mathrm{TG}(\Sigma)$ we denote the class of all root-connected term graphs over Σ. By a 'term graph' we will mean by default a 'root-connected term graph'.

For a Σ-term-graph G and a vertex v of G we denote by $G|_v$ the *sub-term-graph of* G *at* v, that is, the (root-connected) term graph with root v that consists of all vertices that are reachable from v in G. As a useful notation for referring to edges in a term graph G, we will write $v \rightarrowtail_i w$ to indicate that the $(i+1)$-th outgoing edge from vertex v leads to vertex w (that is, $args(v)(i) = w$ with $args$ the argument function of G).

A *rooted* ARS is the extension of an abstract rewriting system (ARS) $\to$ by specifying one of its objects as designated *root*. A rooted ARS $\to$ with objects A and root a is called a *tree* if $\to$ is acyclic (there is no $x \in A$ such that $x \to^+ x$), co-deterministic (for every $x \in A$ there is at most one step of $\to$ with target x), and root-connected (every element $x \in A$ is reachable from a via a sequence of steps of $\to$, i.e. $a \to^* x$).

2 Nested term graphs

We will use the words 'nested' and 'nesting' here in a meaning derived from that of the verb 'nest', which a dictionary[6] explains as 'to fit compactly together or within one another', and as 'to form a hierarchy, series, or sequence of with each member, element, or set contained in or containing the next ⟨nested subroutines⟩'.

A *signature for nested term graphs* (an *ntg-signature*) is a signature Σ for term graphs that is partitioned into a part Σ_{at} for *atomic* symbols, and a part Σ_{ne} for *nested* symbols (cf. the terminals and non-terminals of a context-free string grammar.), that is, $\Sigma = \Sigma_{\mathrm{at}} \cup \Sigma_{\mathrm{ne}}$ and $\Sigma_{\mathrm{at}} \cap \Sigma_{\mathrm{ne}} = \varnothing$. In addition to a given signature Σ for nested term graphs we always assume additional *interface symbols* from the set $OI = O \cup I$, where $O = \{\mathsf{o}\}$ consists of a single unary *output* symbol (symbolizing an edge that can pass on produced output from the root of the term graph definition of a nested symbol), and $I = \{\mathsf{i}_1, \mathsf{i}_2, \mathsf{i}_3, \ldots\}$ is a countably infinite set of *input* symbols with arity zero (symbolizing edges to which input can be supplied to leaves of the term graph definition of a nested symbol).

Definition 1 (recursive specifications for nested term graphs)**.** Let Σ be a signature for nested term graphs. A *recursive (nested term) graph specification* (an *rgs)* over Σ is a tuple $\langle rec, r\rangle$, where:

[6] Merriam-Webster (http://www.merriam-webster.com/dictionary/nest), visited on March 29, 2015.

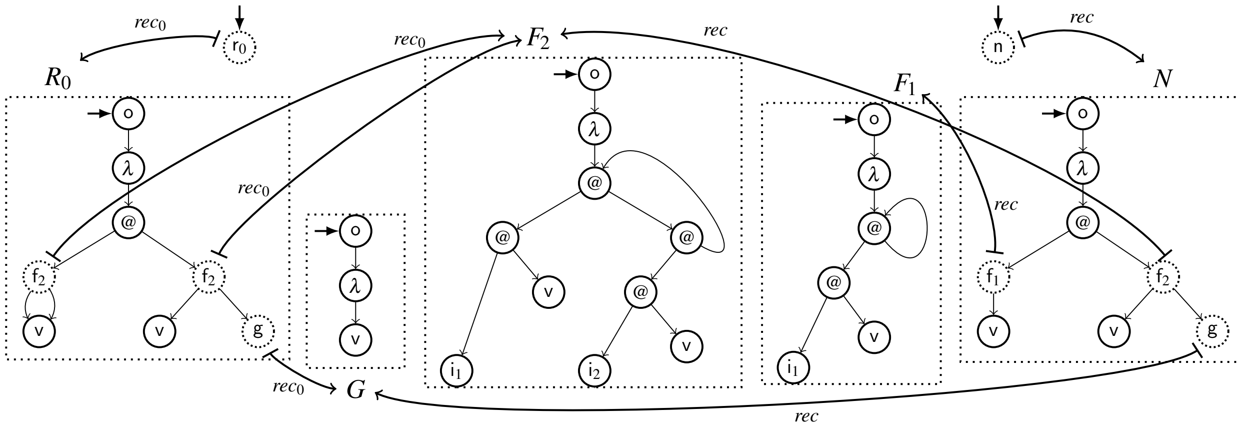


Figure 3: Definitions of a recursive graph specification $\mathcal{R}_0$ (Ex. 2), and a nested term graph $\mathcal{N}$ (Ex. 4).

– $rec : \Sigma_{ne} \to \mathrm{TG}(\Sigma \cup OI)$ is the *specification function* that maps a nested function symbol $f \in \Sigma_{ne}$ with $ar(f) = m$ to a term graph $rec(f) = F \in \mathrm{TG}(\Sigma \cup \{\mathsf{o}, \mathsf{i}_1, \ldots, \mathsf{i}_m\})$ that has precisely one vertex labeled by o, the root, and that contains precisely one vertex labeled by i_j, for each $j \in \{1, \ldots, m\}$;
– $r \in \Sigma_{ne}$, a nullary symbol (that is, $ar(r) = 0$), is the *root symbol*.

For such an rgs $\mathcal{R} = \langle rec, r \rangle$ over Σ, the rooted *dependency* ARS $\multimap$ of $\mathcal{R}$ has as objects the nested symbols in Σ_{ne}, it has root r, and the following steps: for all $f, g \in \Sigma_{ne}$ such that a vertex labeled by g occurs in the term graph $rec(f)$ at position p there is a step $p : f \multimap g$. We say that an rgs $\mathcal{R}$ is *root-connected* if every nested symbol is reachable from the root symbol of $\mathcal{R}$ via steps of the dependency ARS $\multimap$ of $\mathcal{R}$. Analogously as for term graphs, by an ‘rgs’ we will by default mean a ‘root-connected ARS’.

Example 2. We choose a signature part $\Sigma_{at} = \{\lambda/1, @/2, \mathsf{v}/0\}$ for expressing λ-terms as term graphs.

(i) Let $\Sigma_{0,ne} = \{\mathsf{r}_0/0, \mathsf{f}_2/2, \mathsf{g}/0\}$. Then $\mathcal{R}_0 = \langle rec_0, \mathsf{r}_0 \rangle$, where $rec_0 : \Sigma_{0,ne} \to \mathrm{TG}(\Sigma \cup OI)$ is defined by $\mathsf{r}_0 \mapsto R_0$, $\mathsf{f}_2 \mapsto F_2$, and $\mathsf{g} \mapsto G$ as shown in Figure 3 (starting from r_0 on the left), is an rgs.

(ii) Let $\Sigma_{ne} = \{\mathsf{n}/0, \mathsf{f}_1/1, \mathsf{f}_2/2, \mathsf{g}/0\}$. Then $\langle rec, \mathsf{n} \rangle$, where $rec : \Sigma_{ne} \to \mathrm{TG}(\Sigma \cup OI)$ is defined by $\mathsf{n} \mapsto N$, $\mathsf{f}_1 \mapsto F_1$, $\mathsf{f}_2 \mapsto F_2$, and $\mathsf{g} \mapsto G$ as shown in Figure 3 (starting from n on the right), is an rgs. It is an rgs-representation of the gletrec-expression left in Figure 2.

(iii) Let $\Sigma_{1,ne} = \{\mathsf{f}/0, \mathsf{g}/1\}$. Then $\mathcal{R}_1 = \langle rec_1, \mathsf{f} \rangle$, where $rec_1 : \Sigma_{1,ne} \to \mathrm{TG}(\Sigma \cup OI)$ is defined by $\mathsf{f} \mapsto F$, $\mathsf{g} \mapsto G$ as shown left in Figure 4 is an rgs. It represents the gletrec-expression right in Figure 2.

(iv) Let $\Sigma_{2,ne} = \{\mathsf{f}\} \cup \{\mathsf{g}_i/1 \mid i \in \mathbb{N}, i \geq 1\}$. Then $\langle rec_2, \mathsf{f} \rangle$, where $rec_2 : \Sigma_{1,ne} \to \mathrm{TG}(\Sigma \cup OI)$ is defined by $\mathsf{f} \mapsto F$, $\mathsf{g}_1 \mapsto G_1$, $\mathsf{g}_2 \mapsto G_2$, $\mathsf{g}_3 \mapsto G_3$, ... as shown right in Figure 4 is an rgs. It represents the infinite λ-term to the left of it in Figure 4.

Definition 3 (nested term graphs)**.** Let Σ be an ntg-signature. A *nested term graph* (an *ntg*) over Σ is an rgs $\mathcal{N} = \langle rec, r \rangle$ such that the rooted dependency ARS $\multimap$ is a tree. By $\mathcal{NG}(\Sigma)$ we denote the class of all nested term graphs over Σ.

Example 4. We first consider the rgs $\mathcal{R}_0 = \langle rec_0, \mathsf{r}_0 \rangle$ from Example 2, (i). Its rooted dependency ARS $\multimap$ is not a tree, because there are two steps that witness $\mathsf{r}_0 \multimap \mathsf{f}_2$, namely those that are induced by the two occurrences of f_2 in the term graph $R_0 = rec_0(\mathsf{r}_0)$. As a consequence, $\mathcal{R}_0$ is not a nested term graph.

Similarly, the rgs $\mathcal{R}_1$ from Example 2, (iii), is not a nested term graph, because its dependency ARS $\multimap$ contains the cycle $\mathsf{g} \multimap \mathsf{g}$, and hence is not a tree.

But for the rgs $\mathcal{N} = \langle rec, \mathsf{n} \rangle$ from Example 2, (ii), we find that the rooted dependency ARS $\multimap$ is a tree with root n. Hence $\mathcal{N}$ is a nested term graph. For a ‘pretty print’ of $\mathcal{N}$, see the left graph in Figure 3.

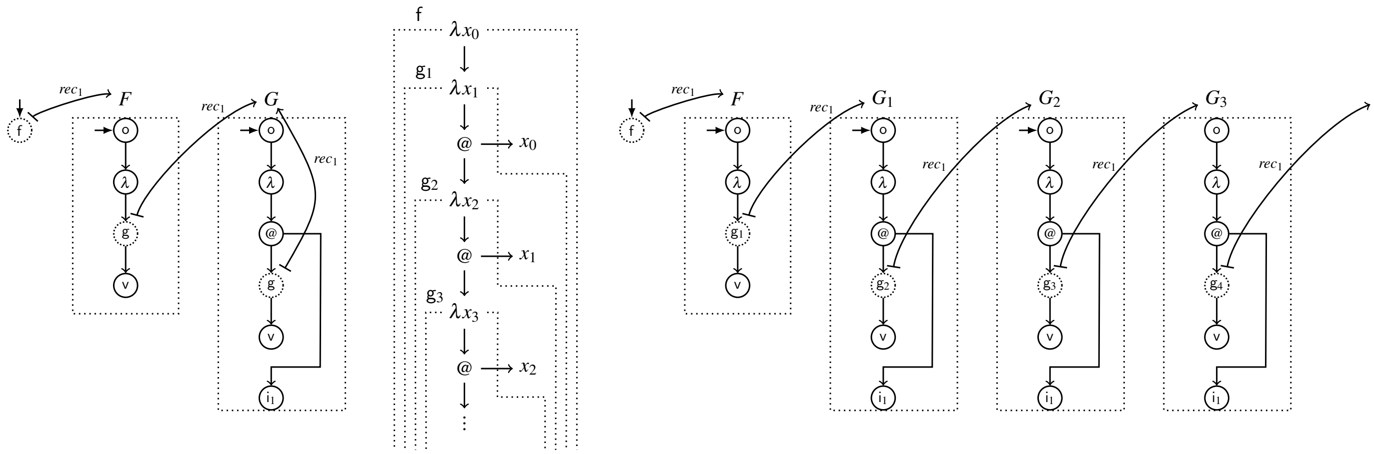


Figure 4: Illustrations of a recursive graph specification $\mathcal{R}_1$ (left, see Example 2, (iii)) with cyclic dependency ARS, and a nested term graph $\mathcal{N}_1$ (right, see Example 2, (iv)) with infinite dependency ARS. Both represent the infinite λ-term (in between them) with infinite nesting of extended scopes.

Also for the rgs $\mathcal{N}_2 = \langle rec_2, \mathsf{f}\rangle$ from Example 2, (iv), we find that the rooted dependency ARS $\multimapinv$ is a tree with root f, since it is of the form: $\mathsf{f} \multimapinv \mathsf{g}_1 \multimapinv \mathsf{g}_2 \multimapinv \mathsf{g}_3 \multimapinv \ldots$. Hence $\mathcal{N}_2$ is a nested term graph with infinitely deep nesting. It represents the infinite λ-term with infinitely deep nesting of its 'extended scopes' (minimal extensions of bound variable scopes in order to obtain nestedness) to the left of it in Figure 4, which has the gletrec-representation on the right in Figure 2.

Next to nested term graphs as functional representations, we also introduce corresponding representations of ntgs as enrichments of ordinary term graphs. The reason is threefold. We obtain a characterization of nested term graphs as integral graph structures with functional dependencies represented by explicit links (see Proposition 7). Furthermore, such structural representations directly induce a behavioral semantics via the associated notions of homomorphism and bisimulation (see Section 3). And finally, they will be instrumental in defining the interpretation of nested term graphs as first-order term graphs (in Section 4).

In 'structural representations' of nested term graphs as defined below, the device of the 'ancestor function' records, and—due to appropriate conditions on it—guarantees, the nesting structure of vertices by assigning to every vertex v the word $anc(v) = v_1 \cdots v_n$ made up of the vertices in which v is nested.

Definition 5 (nested term graphs, as structures)**.** Let Σ be a signature for nested term graphs. A *structural representation* of a nested term graph (an *sntg*) over Σ is a tuple $\langle V, lab, args, call, return, anc, root\rangle$, where $G_0 = \langle V, lab, args, root\rangle$ is a (typically not root-connected) term graph over $\Sigma \cup OI$, and additionally:

- $call : V \rightharpoonup V$ is the *call* (or *step-into*) partial function that assigns to every vertex v labeled with a nested symbol the root of the term graph nested into v (this root is an output vertex);
- $return : V \rightharpoonup V$ is the *return* (or *step-out*) partial function that to every input vertex v labeled by i_j assigns the j-th successor of the vertex into which the term graph containing v is nested;
- $anc : V \to V^*$ is the *ancestor function* that to every vertex v assigns the word $anc(v) = v_1 \cdots v_n$ made up of the vertices in which v is nested: v is nested in v_n, v_n is nested in v_{n-1}, ..., v_2 is nested in v_1;

that satisfy, more precisely, the following conditions, for all $i, k \in \mathbb{N}$, and all $w, w_i, v_1, \ldots, v_k \in V$:

$$(\text{root})_{lab,anc} \qquad lab(root) \in \Sigma_{\text{ne}} \;\wedge\; anc(root) = \varepsilon$$

$$(\text{nested})_{anc} \qquad anc(w) = v_1 \cdots v_n \implies v_1, \ldots, v_n, \text{ and } w \text{ are distinct}$$

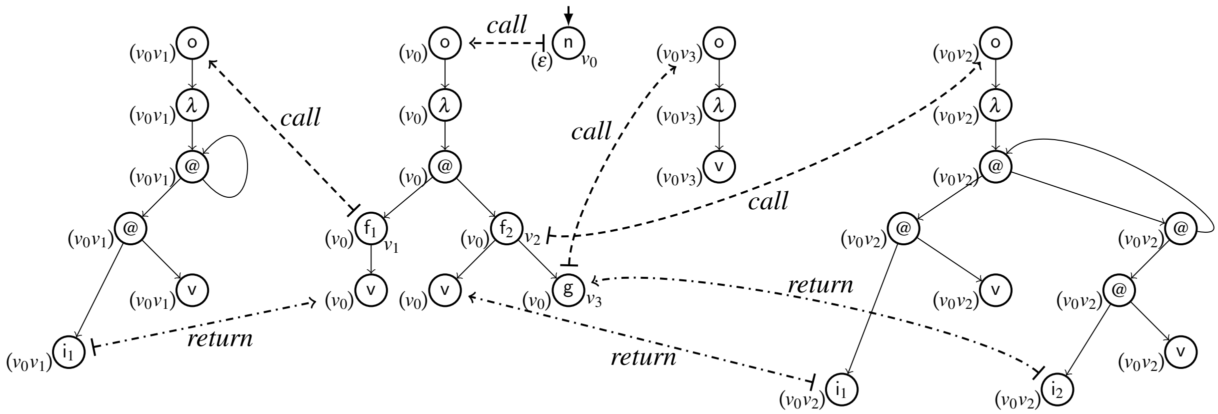


Figure 5: Illustration of a structural representation of the nested term graph $\mathcal{N}$ from Ex. 4, with names for vertices with nested symbols (right of such vertices), and the ancestor function values indicated in brackets.

$$
\begin{array}{rl}
(\text{arguments})_{anc} & w \rightarrowtail_i w_i \implies anc(w_i) = anc(w) \\
(\text{defined})_{call,return} & (\, call(w)\!\downarrow \iff lab(w) \in \Sigma_{\text{ne}} \,) \wedge (\, return(w)\!\downarrow \iff lab(w) \in I \,) \\
(\text{step-into})_{call} & lab(w) \in \Sigma_{\text{ne}} \implies \begin{cases} lab(call(w)) = \mathsf{o} \in O \;\wedge\; anc(call(w)) = anc(w) \cdot w \\ \wedge\; call(w) \text{ is the single vertex with label } \mathsf{o} \text{ in the} \\ \quad \text{sub-term-graph } G_0|_{call(w)} \text{ of } G_0 \text{ at vertex } call(w) \end{cases} \\
(\text{step-out})_{return} & lab(w) \in \Sigma_{\text{ne}} \implies \begin{cases} \text{for all } j \in \{1,\ldots,ar(lab(w))\},\ G_0|_{call(w)} \text{ contains precisely} \\ \text{one vertex } w'_j \text{ with label } \mathsf{i}_j \in I \text{, and it holds: } w \rightarrowtail_j return(w'_j); \\ G_0|_{call(w)} \text{ has no other vertices with labels in } I \end{cases}
\end{array}
$$

Example 6. An sntg that corresponds to the nested term graph $\mathcal{N}$ in Example 4 is depicted in Figure 5.

Proposition 7. *Every nested term graph has a structural representation. And for every structural representation $\mathcal{G}$ of a nested term graph there is a nested term graph for which $\mathcal{G}$ is the structural representation.*

3 Bisimulation and nested bisimulation

In order to motivate appropriate definitions of behavioral semantics for nested term graphs and recursive graph specifications, we start with the rather clear behavioral semantics for sntg's. Then we adapt these definitions to nested term graphs, and yield corresponding concepts. Subsequently we develop a definition of homomorphism and bisimilarity that also applies to recursive graph specifications, and is based on purely local progression rules together with stacks that record the nesting history. We call these further concepts 'nested homomorphism' and 'nested bisimilarity'. Finally we gather statements that relate bisimilarity and nested bisimilarity.

Homomorphisms and bisimulations between sntg's. Since structural representations of nested term graphs can be viewed as coalgebras, they carry clear associated notions of homomorphism and bisimilarity. To see this, let $\mathcal{G}_1$ and $\mathcal{G}_2$ be sntg's over signatures Σ_1 and Σ_2 with the same part Σ_{at} for atomic symbols. A homomorphism from $\mathcal{G}_1$ to $\mathcal{G}_2$ (indicated by $\mathcal{G}_1 \mathrel{\underline{\rightarrow}} \mathcal{G}_2$) is a function $\phi : V_1 \to V_2$ between their vertex sets that preserves the property of being root, preserves atomic, nested, and interface labels, commutes with the partial functions *call* and *return*, commutes with the (individual) argument function on vertices with atomic labels, and preserves the ancestor function. A bisimulation between sntg's $\mathcal{G}_1$ and $\mathcal{G}_2$ can then be defined as an sntg $\mathcal{G}$ with the property $\mathcal{G}_1 \mathrel{\underline{\leftarrow}} \mathcal{G} \mathrel{\underline{\rightarrow}} \mathcal{G}_2$.

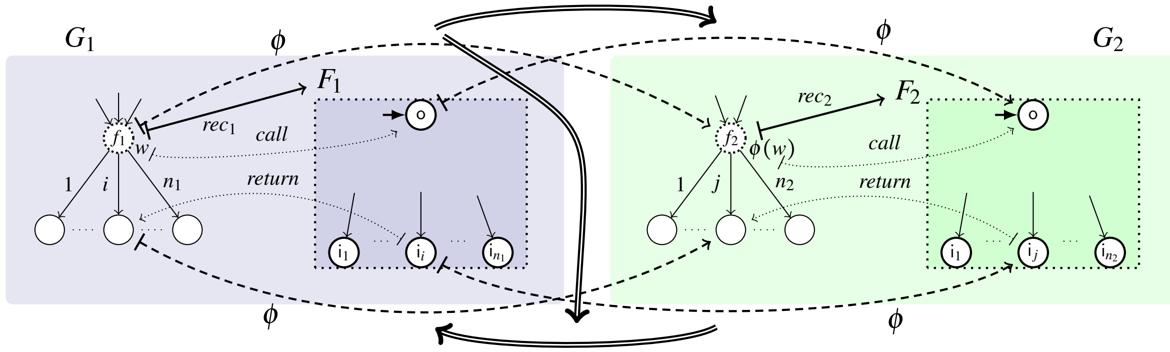


Figure 6: Bisimulation 'interface clause' for homorphisms between ntgs in case of related vertices with nested symbols. Its motivation consists in following the dotted *call*- and *return*-links of the corresponding sntg's: if ϕ maps a vertex w with nested symbol f_1 to vertex $\phi(w)$ with nested symbol f_2, then ϕ must also map the root of the definition F_1 of f_1 to the root of the definition F_2 of f_2; and if ϕ maps the input vertex i_i of F_1 to an input vertex i_j of F_2, then ϕ must also map the i-th successor of w to the j-th successor of $\phi(w)$.

Definition 8 (homomorphism, bisimulation between sntg's)**.** Let $\Sigma_1 = \Sigma_{\mathrm{at}} \cup \Sigma_{1,\mathrm{ne}}$ and $\Sigma_2 = \Sigma_{\mathrm{at}} \cup \Sigma_{2,\mathrm{ne}}$ be ntg-signatures with the same signature Σ_{at} for atomic symbols. Furthermore, let for each of $i \in \{1,2\}$, $\mathcal{G}_i = \langle V_i, lab_i, args_i, anc_i, call_i, return_i, root_i \rangle$ be an sntg over signature Σ.

A *homomorphism* (functional bisimulation) between $\mathcal{G}_1$ and $\mathcal{G}_2$ is a $\langle \Sigma_{\mathrm{ne}}, \Sigma_{\mathrm{at}}, O, I \rangle$-respecting morphism $\phi : V_1 \to V_2$ between the structures $\mathcal{G}_1$ and $\mathcal{G}_2$, that is, for all $w \in V_1$ the following conditions hold:

$$
\begin{array}{rcll}
 & \Longrightarrow & \phi(root_1) = root_2 \;\wedge\; \phi^*(anc_1(w)) = anc_2(\phi(w)) & (root, anc) \\
lab_1(w) \in \Sigma_{\mathrm{at}} & \Longrightarrow & lab_2(\phi(w)) = lab_1(w) \in \Sigma_{\mathrm{at}} \;\wedge\; \phi^*(args_1(w)) = args_2(\phi(w)) & (lab, args)_{\Sigma_{\mathrm{at}}} \\
lab_1(w) \in \Sigma_{1,\mathrm{ne}} & \Longrightarrow & lab_2(\phi(w)) \in \Sigma_{2,\mathrm{ne}} \;\wedge\; \phi(call_1(w)) = call_2(\phi(w)) & (lab, call)_{\Sigma_{\mathrm{ne}}} \\
lab_1(w) \in O & \Longrightarrow & lab_2(\phi(w)) \in O & (lab)_O \\
lab_1(w) \in I & \Longrightarrow & lab_2(\phi(w)) \in I \;\wedge\; \phi(return_1(w)) = return_2(\phi(w)) & (lab, return)_I
\end{array}
$$

where ϕ^* is the homomorphic extension of ϕ to a function from V_1^* to V_2^*. If there is a homomorphism ϕ from $\mathcal{G}_1$ to $\mathcal{G}_2$, we write $\mathcal{G}_1 \mathrel{\underline{\to}}_{\phi} \mathcal{G}_2$ and $\mathcal{G}_2 \mathrel{\underline{\leftarrow}}_{\phi} \mathcal{G}_1$, or, dropping ϕ as subscript, $\mathcal{G}_1 \mathrel{\underline{\to}} \mathcal{G}_2$ and $\mathcal{G}_2 \mathrel{\underline{\leftarrow}} \mathcal{G}_1$.

A *bisimulation* between $\mathcal{G}_1$ and $\mathcal{G}_2$ is an sntg $\mathcal{G} = \langle B, lab, args, call, return, anc, root \rangle$ where $B \subseteq V_1 \times V_2$ and $root = \langle root_1, root_2 \rangle$ such that $\mathcal{G}_1 \mathrel{\underline{\leftarrow}}_{\pi_1} \mathcal{G} \mathrel{\underline{\to}}_{\pi_2} \mathcal{G}_2$ where π_1 and π_2 are projection functions that are defined, for $i \in \{1,2\}$, by $\pi_i : V_1 \times V_2 \to V_i$, $\langle v_1, v_2 \rangle \mapsto v_i$. If there exists a bisimulation B between $\mathcal{G}_1$ and $\mathcal{G}_2$, then we write $\mathcal{G}_1 \mathrel{\underline{\leftrightarrow}} \mathcal{G}_2$, and say that $\mathcal{G}_1$ is *bisimilar to* $\mathcal{G}_2$.

Homomorphisms and bisimulations between nested term graphs. The definitions for sntg's above can motivate similar definitions for ntgs. Let $\mathcal{N}_1 = \langle rec_1, r_1 \rangle$ and $\mathcal{N}_2 = \langle rec_2, r_2 \rangle$ be ntgs over signatures with the same atomic symbols. A homomorphism between $\mathcal{N}_1$ and $\mathcal{N}_2$ will be defined as a function $\phi : V_1 \to V_2$ between the vertex sets of the disjoint unions of the term graphs in the image of rec_1 and rec_2, respectively; on vertices labeled with atomic, or interface labels, ϕ behaves like an ordinary term graph homomorphism; and on vertices labeled with nested symbols an 'interface' clause applies. This condition, illustrated in Figure 6, demands that via ϕ related vertices v and w with nested symbols entail, following *call*-links of the corresponding sntg's, that the roots of the symbol definition are related via ϕ, and, following the *return*-links of the corresponding sntg's, that respective successors of v and w are related via ϕ.

Definition 9 (homomorphism, bisimulation between ntgs)**.** Let $\Sigma_1 = \Sigma_{\mathrm{at}} \cup \Sigma_{1,\mathrm{ne}}$ and $\Sigma_2 = \Sigma_{\mathrm{at}} \cup \Sigma_{2,\mathrm{ne}}$ be ntg-signatures with the same signature Σ_{at} for atomic symbols. Let $\mathcal{N}_1 = \langle rec_2, r_1 \rangle$ and $\mathcal{N}_2 = \langle rec_2, r_2 \rangle$ be nested term graphs over Σ_1 and Σ_2, respectively. Let $G_i = \langle V_i, lab_i, args_i, rt_i, ins_i \rangle$ for $i \in \{1,2\}$ be the enriched (not necessarily root-connected) term graphs that arise as the disjoint union of the term graphs $rec_i(f)$ for $f \in \Sigma_{i,\mathrm{ne}}$ together with functions $rt_i : \Sigma_{i,\mathrm{ne}} \to V_i$ and $ins_i : \Sigma_{i,\mathrm{ne}} \to \wp(V_i)$ that map a nested function symbol f to the root $rt_i(f)$, and to the set $ins_i(f)$ of input vertices, of the definition of f in G_i, respectively.

A *homomorphism* between $\mathcal{N}_1$ and $\mathcal{N}_2$ is a function $\phi : V_1 \to V_2$ such that for all $w \in V_1$ it holds (a condition in brackets $[\ldots]$ has been added for clarity, but is redundant, see Remark 10):

$$\phi(rt_1(r_1)) = rt_2(r_2) \qquad \text{(roots of rgs's)}$$

$$lab_1(w) \in \Sigma_{\mathrm{at}} \implies lab_2(\phi(w)) = lab_1(w) \in \Sigma_{\mathrm{at}} \;\wedge\; \phi^*(args_1(w)) = args_2(\phi(w)) \qquad (lab, args)_{\Sigma_{\mathrm{at}}}$$

$$lab_1(w) \in O \implies lab_2(\phi(w)) \in O \qquad (lab)_O$$

$$lab_1(w) \in I \implies lab_2(\phi(w)) \in I \qquad (lab)_I$$

$$lab_1(w) \in \Sigma_{1,\mathrm{ne}} \implies \begin{cases} lab_2(\phi(w)) \in \Sigma_{2,\mathrm{ne}} \;\wedge\; \phi(rt_1(lab_1(w))) = rt_2(lab_2(\phi(w))) \\ [\;\wedge\; \forall u \in ins_1(lab_1(w)).\; \phi(u) \in ins_2(lab_2(\phi(w)))\;] \\ \wedge\; \forall u \in ins_1(lab_1(w)). \forall i, j \in \mathbb{N}. \forall x \in V_1. \forall y \in V_2. \\ \qquad lab_1(u) = \mathsf{i}_i \;\wedge\; lab_2(\phi(u)) = \mathsf{i}_j \;\wedge\; w \rightarrowtail_i x \;\wedge\; \phi(w) \rightarrowtail_j y \\ \qquad\qquad \Rightarrow\; \phi(x) = y \end{cases} \qquad (lab, args)_{\Sigma_{\mathrm{ne}}}$$

hold, where ϕ^* is the homomorphic extension of ϕ to a function from V_1^* to V_2^*. See Figure 6 for an illustration of the 'interface clause' $(lab, args)_{\Sigma_{\mathrm{ne}}}$. If there is a homomorphism ϕ from $\mathcal{N}_1$ to $\mathcal{N}_2$, we write $\mathcal{N}_1 \mathrel{\underline{\rightarrow}}_\phi \mathcal{N}_2$ and $\mathcal{N}_2 \mathrel{\underline{\leftarrow}}_\phi \mathcal{N}_1$, or, dropping ϕ as subscript, $\mathcal{N}_1 \mathrel{\underline{\rightarrow}} \mathcal{N}_2$ and $\mathcal{N}_2 \mathrel{\underline{\leftarrow}} \mathcal{N}_1$.

A *bisimulation* between $\mathcal{N}_1$ and $\mathcal{N}_2$ is an ntg $\mathcal{N}$ over signature $\Sigma = \Sigma_{\mathrm{at}} \cup \Sigma_{\mathrm{ne}}$ with $\Sigma_{\mathrm{ne}} \subseteq \Sigma_{1,\mathrm{ne}} \times \Sigma_{2,\mathrm{ne}}$ such that $\mathcal{N}_1 \mathrel{\underline{\leftarrow}}_{\langle \pi_1, \phi \rangle} \mathcal{N} \mathrel{\underline{\rightarrow}}_{\langle \pi_2, \phi \rangle} \mathcal{N}_2$ where π_1 and π_2 are projection functions, defined, for $i \in \{1,2\}$, by $\pi_i : \Sigma_{1,\mathrm{ne}} \times \Sigma_{2,\mathrm{ne}} \to \Sigma_{i,\mathrm{ne}}$, $\langle f_1, f_2 \rangle \mapsto f_i$.

Remark 10. In condition $(lab, args)_{\Sigma_{\mathrm{ne}}}$ for a homomorphism between nested term graphs in Definition 9 the part $\forall u \in ins_1(lab_1(w)).\; \phi(u) \in ins_2(lab_2(\phi(w)))$ is redundant. It expresses that if a homomorphism ϕ maps the root $rt_1(\mathsf{f}_1)$ in G_1 of the definition a nested function symbol f_1 to the root $rt_2(\mathsf{f}_2)$ in G_2 of the definition of a nested function symbol f_2 (by Definition 1, $rt_1(\mathsf{f}_1)$ and $rt_2(\mathsf{f}_2)$ must be output vertices), then ϕ maps input vertices of the definition of f_1 in G_1 to input vertices of the definition of f_2 in G_2. This, and additionally also the fact that $\forall x \in ins_2(lab_2(\phi(w)))\, \exists u \in ins_1(lab_1(w)).\; \phi(u) = x$ holds, follow from the other conditions since a homomorphism is a function, and importantly, since definitions of nested symbols are term graphs that were assumed to be root-connected by default. By the latter, input vertices of the definition of a nested symbol are always reachable from the output vertex at the root of the definition, which facilitates a proof of these properties using induction on the length of paths from output to input vertices.

Example 11. See Figure 7 for four nested term graphs that are related by homomorphisms, and hence are bisimilar. Note that homomorphisms can map a nested symbol to one of smaller arity (here from arity 2 to arity 1). For the nested term graphs $\mathcal{N}$ and $\mathcal{N}(\mathcal{R})$ in Figure 8 (the notation $\mathcal{N}(\mathcal{R})$ will become clear later in Definition 15) it holds that $\mathcal{N}(\mathcal{R}) \mathrel{\underline{\rightarrow}} \mathcal{N}$, and hence that they are bisimilar; but there is no homomorphism from $\mathcal{N}$ to $\mathcal{N}(\mathcal{R})$, and hence $\mathcal{N} \mathrel{\underline{\not\rightarrow}} \mathcal{N}(\mathcal{R})$.

Proposition 12. *The notions of homomorphism and bisimilarity for ntgs correspond to the notions of homomorphism and bisimilarity for sntg's, via the mappings between these concepts stated in Proposition 7.*

Nested bisimulation and nested homomorphism between rgs's and ntgs. A 'nested bisimulation' compares ntgs, or for that matter also rgs's, by keeping track, along any chosen path, of the nesting history by

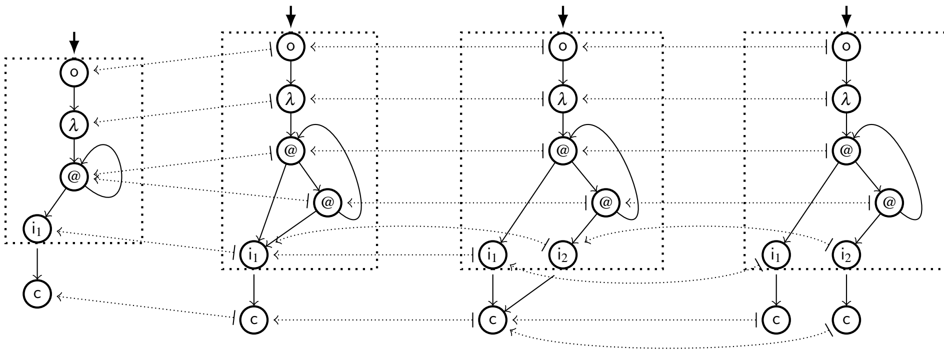


Figure 7: Four simple nested term graphs that are related by converse functional bisimilarity $\leftarrow\!\!\!\!\!-$ (and hence also by bisimilarity $\underline{\leftrightarrow}$) via homorphisms that are indicated as dotted assignments.

means of stacks of nested vertices. It is defined between prefixed expressions $(v_1 \cdots v_k)v$ and $(w_1 \cdots w_k)w$ that describe a visit of the vertices v and w in the context of histories of visits to vertices v_i and w_i as recorded by the stacks $v_1 \cdots v_k$ and $w_1 \cdots w_k$ of the nested vertices in the nesting hierarchy above v and w, respectively. These stacks facilitate the definition of nested bisimulation by purely local progression rules, since the immediate nesting ancestor of a vertex can always be found on top of the stack.

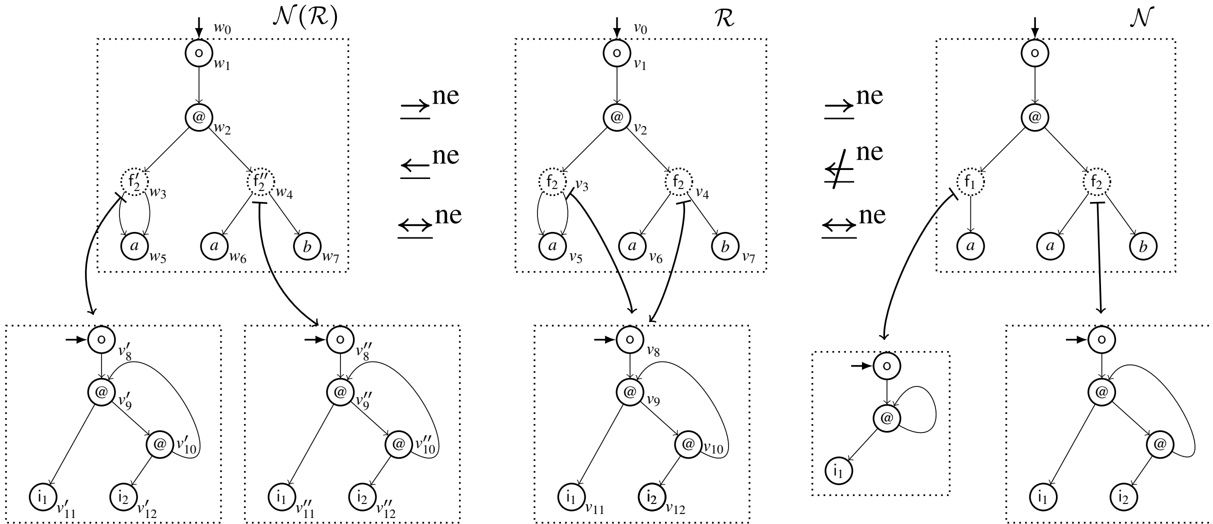
Definition 13 (nested bisimulation and nested homomorphism between rgs's and ntgs)**.** Let $\Sigma_1 = \Sigma_{\text{at}} \cup \Sigma_{1,\text{ne}}$ and $\Sigma_2 = \Sigma_{\text{at}} \cup \Sigma_{2,\text{ne}}$ be ntg-signatures with the same signature Σ_{at} for atomic symbols. Let $\mathcal{R}_1 = \langle rec_2, r_1 \rangle$ and $\mathcal{R}_2 = \langle rec_2, r_2 \rangle$ be rgs's over Σ_1 and Σ_2, respectively. Let $G_i = \langle V_i, lab_i, args_i, root_i, rt_i \rangle$ for $i \in \{1,2\}$ be the enriched (typically not root-connected) term graph that arises as the disjoint union of the term graphs $rec_i(f)$ for $f \in \Sigma_{i,\text{ne}}$ such that its root $root_i \in V_i$ is the root of $rec_i(r_i)$, and with as enrichment the function $rt_i : \Sigma_{i,\text{ne}} \to V_i$ that maps a nested function symbol $f \in \Sigma_{i,\text{ne}}$ to its root $rt_i(f)$ in G_i (hence $root_i = rt_i(r_i)$).

A *nested bisimulation* between $\mathcal{R}_1$ and $\mathcal{R}_2$ is a relation $B_{\text{ne}} \subseteq V_1^* \times V_1 \times V_2^* \times V_2$, for which we will indicate elements $\langle v_1 \cdots v_k, v, w_1 \cdots w_k, w \rangle \in B_{\text{ne}}$ as $(v_1 \cdots v_k)v \; B_{\text{ne}} \; (w_1 \cdots w_k)w$, with the following properties, for all $i, j, k \in \mathbb{N}$, $v, v_1, \ldots, v_k, v_i' \in V_1$, $w, w_1, \ldots, w_k, w_i', w_j' \in V_2$, $f \in \Sigma_{\text{at}}$, $f_1 \in \Sigma_{1,\text{ne}}$, and $f_2 \in \Sigma_{2,\text{ne}}$:

$$
\begin{array}{rl}
(root)^{\text{ne}} & ()root_1 \; B_{\text{ne}} \; ()root_2 \quad (\text{equivalently: } ()rt_1(r_1) \; B_{\text{ne}} \; ()rt_2(r_2)) \\
(lab)^{\text{ne}} & (v_1 \cdots v_k)v \; B_{\text{ne}} \; (w_1 \cdots w_k)w \implies (\, lab(v) = lab(w) \in \Sigma_{\text{at}} \,) \vee (\, lab(v) \in \Sigma_{1,\text{ne}} \wedge lab(w) \in \Sigma_{2,\text{ne}} \,) \\
 & \qquad\qquad \vee (\, lab(v) = lab(w) = \mathsf{o} \in O \,) \vee (\, lab(v) \in I \wedge lab(w) \in I \,) \\
(args)^{\text{ne}}_{\Sigma_{\text{at}}} & (v_1 \cdots v_k)v \; B_{\text{ne}} \; (w_1 \cdots w_k)w \;\wedge\; lab_1(v) = lab_2(w) = f \in \Sigma_{\text{at}} \;\wedge\; v \rightarrowtail_i v_i' \;\wedge\; w \rightarrowtail_i w_i' \\
 & \implies (v_1 \cdots v_k)v_i' \; B_{\text{ne}} \; (w_1 \cdots w_k)w_i' \\
(args)^{\text{ne}}_{\Sigma_{\text{ne}}} & (v_1 \cdots v_k)v \; B_{\text{ne}} \; (w_1 \cdots w_k)w \;\wedge\; lab_1(v) = f_1 \in \Sigma_{1,\text{ne}} \;\wedge\; lab_2(w) = f_2 \in \Sigma_{2,\text{ne}} \\
 & \implies (v_1 \cdots v_k v)rt_1(f_1) \; B_{\text{ne}} \; (w_1 \cdots w_k w)rt_2(f_2) \\
(args)^{\text{ne}}_{O} & (v_1 \cdots v_k)v \; B_{\text{ne}} \; (w_1 \cdots w_k)w \;\wedge\; lab_1(v) = lab_2(w) = \mathsf{o} \in O \;\wedge\; v \rightarrowtail_0 v_0' \;\wedge\; w \rightarrowtail_0 w_0' \\
 & \implies (v_1 \cdots v_k)v_0' \; B_{\text{ne}} \; (w_1 \cdots w_k)w_0' \\
(args)^{\text{ne}}_{I} & (v_1 \cdots v_k)v \; B_{\text{ne}} \; (w_1 \cdots w_k)w \;\wedge\; lab_1(v) = \mathsf{i}_i \in I \;\wedge\; lab_2(w) = \mathsf{i}_j \in I \\
 & \implies \exists v_i' \in V_1. \exists w_j' \in V_2. \big(v_k \rightarrowtail_i v_i' \;\wedge\; w_k \rightarrowtail_j w_j' \;\wedge\; (v_1 \cdots v_{k-1})v_i' \; B_{\text{ne}} \; (w_1 \cdots w_{k-1})w_j' \big)
\end{array}
$$

If there is a nested bisimulation B_{ne} between $\mathcal{R}_1$ and $\mathcal{R}_2$, then we write $\mathcal{R}_1 \;\underline{\leftrightarrow}^{\text{ne}}_{B_{\text{ne}}}\; \mathcal{R}_2$, or just $\mathcal{R}_1 \;\underline{\leftrightarrow}^{\text{ne}}\; \mathcal{R}_2$.

A *nested homomorphism* from $\mathcal{R}_1$ to $\mathcal{R}_2$ is a partial function $\phi_{\text{ne}} : V_1^* \times V_1 \rightharpoonup V_2^* \times V_2$ such that the relation $\{\langle v_1 \cdots v_n, v, w_1 \cdots w_n, w \rangle \in V_1^* \times V_1 \times V_2^* \times V_2 \mid \phi_{\text{ne}}(\langle v_1 \cdots v_n, v \rangle){\downarrow} = \langle w_1 \cdots w_n, w \rangle\}$ is a nested bisimulation between $\mathcal{R}_1$ and $\mathcal{R}_2$. If there is such a function ϕ_{ne}, we write $\mathcal{R}_1 \;\underline{\rightarrow}^{\text{ne}}_{\phi_{\text{ne}}}\; \mathcal{R}_2$, or just $\mathcal{R}_1 \;\underline{\rightarrow}^{\text{ne}}\; \mathcal{R}_2$.



$$B_{\text{ne}} = \{\ \langle ()v_0, ()w_0\rangle, \langle (v_0)v_1, (w_0)w_1\rangle, \langle (v_0)v_2, (w_0)w_2\rangle, \langle (v_0)v_3, (w_0)w_3\rangle, \ldots, \langle (v_0)v_7, (w_0)w_7\rangle,$$
$$\langle (v_0v_3)v_8, (w_0w_3)w'_8\rangle, \langle (v_0v_3)v_9, (w_0w_3)w'_9\rangle, \ldots, \langle (v_0v_3)v_{12}, (w_0w_3)w'_{12}\rangle,$$
$$\langle (v_0v_4)v_8, (w_0w_4)w''_8\rangle, \langle (v_0v_4)v_9, (w_0w_4)w''_9\rangle, \ldots, \langle (v_0v_4)v_{12}, (w_0w_4)w''_{12}\rangle\ \}$$

Figure 8: Example of a recursive graph specifications $\mathcal{R}$ (middle), the nested term graph $\mathcal{N}(\mathcal{R})$ specified by $\mathcal{R}$ (left), and a bisimilar nested term graph $\mathcal{N}$ (right) that are related by $\rightharpoonup^{\text{ne}}$ and $\leftrightarrow^{\text{ne}}$ (i.p., $\mathcal{R} \rightharpoonup^{\text{ne}}_{B_{\text{ne}}} \mathcal{N}(\mathcal{R})$).

Example 14. The set B_{ne} defined in Figure 8 is a nested homomorphism from the rgs $\mathcal{R}$ to the nested term graph $\mathcal{N}(\mathcal{R})$ (the notation $\mathcal{N}(\mathcal{R})$ is explained in Definition 15 below). Hence it witnesses $\mathcal{R} \rightharpoonup^{\text{ne}}_{B_{\text{ne}}} \mathcal{N}(\mathcal{R})$. Note that its converse also is a nested homomorphism, and hence that $\mathcal{N}(\mathcal{R}) \rightharpoonup^{\text{ne}} \mathcal{R}$ holds, too. There is also an obvious nested homomorphism from $\mathcal{N}(\mathcal{R})$ to the nested term graph $\mathcal{N}$ in Figure 8 right, but not the other way round, that is, $\mathcal{N} \rightharpoonup^{\text{ne}} \mathcal{N}(\mathcal{R})$ does not hold.

In the example concerning the four nested term graphs in Figure 7, the indicated homomorphisms induce obvious corresponding nested homomorphisms.

Every nested bisimulation B_{ne} between rgs's gives rise to an rgs $\mathcal{R}_{B_{\text{ne}}}$ in a straightforward manner by forming, for every pair $\langle (v_1 \ldots v_n)v, (w_1 \ldots w_n)w\rangle$ a nested function symbol $f_{v_1\ldots v_n, w_1\ldots w_n}$, and by letting the pair be a vertex with label $lab_1(v) = lab_2(w)$ in the term graph specifying $f_{v_1\ldots v_n, w_1\ldots w_n}$. As nesting is recorded in B_{ne}, the rgs $\mathcal{R}_{B_{\text{ne}}}$ turns out to be a nested term graph. Of particular interest are nested self-bisimulations on an rgs, that is, bisimulations between an rgs and itself.

Definition 15 (nested term graph specified by an rgs)**.** Let $\mathcal{R}$ be an rgs over ntg-signature Σ, and B_{ne} the minimal nested bisimulation between $\mathcal{R}$ and itself. Together with $\mathcal{R}$, B_{ne} specifies an rgs $\mathcal{R}_{B_{\text{ne}}}$ with a tree as dependency ARS, and hence an ntg. This is the nested term graph *specified* by $\mathcal{R}$, denoted by $\mathcal{N}(\mathcal{R})$.

Example 16. In Figure 8, the nested term graph $\mathcal{N}(\mathcal{R})$ on the left is specified by the rgs $\mathcal{R}$ in the middle. In Figure 4, the rgs $\mathcal{R}_1$ on the left specifies the nested term graph $\mathcal{N}_1$ on the right, that is, $\mathcal{N}_1 = \mathcal{N}(\mathcal{R}_1)$.

Proposition 17. *For every rgs $\mathcal{R}$ it holds: $\mathcal{N}(\mathcal{R}) \rightharpoonup^{\text{ne}} \mathcal{R}$ and $\mathcal{R} \rightharpoonup^{\text{ne}} \mathcal{N}(\mathcal{R})$.*

Relationships between homomorphism/bisimilarity, and nested homomorphism/nested bisimilarity. We conclude this section with two statements that relate $\rightharpoonup$ and $\leftrightarrow$ with $\rightharpoonup^{\text{ne}}$ and $\leftrightarrow^{\text{ne}}$ on nested term graphs, and nested bisimilarity of rgs's with bisimilarity of the specified nested term graphs.

Theorem 18. *For nested term graphs, functional bisimilarity $\rightleftarrows$ coincides with nested functional bisimilarity $\rightleftarrows^{\mathrm{ne}}$, and bisimilarity $\leftrightarrow$ coincides with nested bisimilarity $\leftrightarrow^{\mathrm{ne}}$.*

The intuition behind this statement is as follows. In building up a bisimulation B between two bisimilar nested term graphs $\mathcal{N}_1$ and $\mathcal{N}_2$, the tree structure of the dependency ARSs $\multimapinv$ together with the interface clause for bisimulation guarantees that a vertex v with nesting ancestors $v_1 \dots v_{k_1}$ is only related to a vertex w with nesting ancestors $w_1 \dots w_{k_2}$ if $k_1 = k_2$. And furthermore, that by adding the nesting ancestors of vertices as prefixes the bisimulation B gives rise to a nested bisimulation B_{ne} between $\mathcal{N}_1$ and $\mathcal{N}_2$. Vice versa, again due to tree structure of the dependency ARSs $\multimapinv$, the contextual information in a nested bisimulation can be ignored to obtain a bisimulation. Formally, Theorem 18 and Theorem 19 below, can be proved by using induction on the length of 'access paths' (acyclic paths from the root to a vertex).

Theorem 19. *Two recursive graph specifications $\mathcal{R}_1$ and $\mathcal{R}_2$ are nested bisimilar (i.e. $\mathcal{R}_1 \leftrightarrow^{\mathrm{ne}} \mathcal{R}_2$) if and only if the nested term graphs specified by $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively, are bisimilar (i.e. $\mathcal{N}(\mathcal{R}_1) \leftrightarrow \mathcal{N}(\mathcal{R}_2)$).*

4 Interpretation as first-order term graphs

Nested term graphs can be interpreted in a faithful, and rather natural way as first-order term graphs. By 'faithful' we mean that the interpretation mapping is a retraction that preserves and reflects homomorphisms, and by 'natural' that it can be defined inductively on the nesting structure. The basic idea is analogous to the interpretation of λ-higher-order-term-graphs as first-order λ-term-graphs developed in [7, 8].

For a nested term graph $\mathcal{N} = \langle rec, r\rangle$ we define the first-order term graph interpretation $I(\mathcal{N})$ of $\mathcal{N}$ by a stepwise procedure that starts on an sntg representation of $\mathcal{N}$ as input. The example of the ntg $\mathcal{N}$ and the resulting interpretation $I(\mathcal{N})$ in Figure 1 may help to provide some guiding intuition.

Definition 20. Let $\mathcal{N}$ be a nested term graph over $\Sigma = \Sigma_{\mathrm{at}} \cup \Sigma_{\mathrm{ne}}$, and $\mathcal{G} = \langle V, lab, args, call, return, anc, root\rangle$ be an sntg representation of $\mathcal{N}$. With $\Sigma_{\mathrm{at}} = \Sigma_{\mathrm{at,const}} \uplus \Sigma_{\mathrm{at,fun}}$ a partitioning of Σ_{at} into constant and non-constant symbols, let $\Sigma_{\mathrm{at,const'}} = \{\mathsf{c}'/1 \mid \mathsf{c} \in \Sigma_{\mathrm{at,const}}\}$ (that is, the constants in Σ_{at} are turned into corresponding unary symbols in $\Sigma_{\mathrm{at,const'}}$), and let $\Sigma' = \Sigma_{\mathrm{at,const'}} \cup \Sigma_{\mathrm{at,fun}} \cup \{\mathsf{o}/1, \mathsf{i}/2, \mathsf{o_r}/1, \mathsf{i_r}/1\}$. The *first-order term graph interpretation $I(\mathcal{N})$ of $\mathcal{N}$* is a term graph over Σ' that is obtained from $\mathcal{G}$ by the following steps:

- (i) Remove every vertex v with a nested symbol, redirect incoming edges at v to the vertex $call(v)$.
- (ii) Relabel every input vertex w with nullary label i_k by the binary label i (thereby dropping the index k), directing the first edge (which becomes a back-link) from w to $return(w)$, and the second edge from w to the vertex $call(v_n)$, where $anc(w) = v_1 \dots v_n$ (note that $call(v_n)$ has label o).
- (iii) Relabel the output vertex (with label o) at the root by the special unary symbol $\mathsf{o_r}$.
- (iv) Change every vertex with a nullary symbol c into a vertex labeled with a corresponding unary symbol c' whose outgoing edge targets a chain of new binary input vertices whose back-links target respective output vertices of the nesting structure; the outermost input vertex gets label $\mathsf{i_r}$ and a backlink to $\mathsf{o_r}$.

Example 21. See Appendix A on pages 17–18 for an application of this procedure on the sntg in Example 4.

The statements that show that this interpretation is indeed faithful are closely analogous to the statements that establish this fact for the interpretation of λ-higher-order-term-graphs by the first-order 'λ-term-graphs' introduced in [7]. Here we only describe the most important steps and their underlying intuition.

The first step is as follows. In analogy with the class λ-term-graphs in [7], those first-order term graphs that arise as interpretations of nested term graphs belong to a class of term graphs that can be defined via the existence of an ancestor function with appropriate properties, see the definition below. The name of this class already anticipates the fact that all of its members do indeed represent nested term graphs.

Definition 22 (term graphs that represent nested term graphs)**.** Let Σ be an ntg-signature, and let Σ' be defined as in Definition 20. Let $G = \langle V, lab, args, root\rangle$ be a term graph, and $anc : V \to V^*$ be a function. We say that G is *correct with respect to ancestor function anc* if for all $w, w_0, w_1 \in V$ and all $i \in \mathbb{N}$ the following conditions hold (conditions in brackets $[\ldots]$ have been added for readability, but are redundant):

$$\begin{aligned}
&\Rightarrow\ lab(root) = \mathsf{o_r} \wedge anc(root) = \varepsilon\\
lab(w) \in \Sigma_{\text{at,const}'} \wedge w \rightarrowtail_0 w_0 &\Rightarrow\ anc(w_0) = anc(w) \wedge \exists n \in \mathbb{N}.\, \exists w_1, \ldots, w_n \in V.\\
&\qquad\qquad \big(w_0 \rightarrowtail_0 w_1 \rightarrowtail_0 \ldots \rightarrowtail_0 w_n\\
&\qquad\qquad\ \wedge lab(w_0) = \ldots = lab(w_{n-1}) = \mathsf{i} \wedge lab(w_n) = \mathsf{i_r} \big)\\
lab(w) \in \Sigma_{\text{at,fun}} \wedge w \rightarrowtail_i w_i &\Rightarrow\ anc(w_i) = anc(w)\\
lab(w) \in \{\mathsf{o_r}, \mathsf{o}\} \wedge w \rightarrowtail_0 w_0 &\Rightarrow\ anc(w_0) = anc(w) \cdot w\\
lab(w) = \mathsf{i_r} \wedge w \rightarrowtail_0 w_0 &\Rightarrow\ [\, root = w_0 =]\ anc(w_0) \cdot w_0 = anc(w) \quad [\wedge\, lab(w_0) = \mathsf{o_r}]\\
lab(w) = \mathsf{i} \wedge w \rightarrowtail_0 w_0 &\Rightarrow\ anc(w_0) \cdot v = anc(w) \text{ for some } v \in V\\
lab(w) = \mathsf{i} \wedge w \rightarrowtail_1 w_1 &\Rightarrow\ anc(w_1) \cdot w_1 = anc(w) \quad [\wedge\, lab(w_1) = \mathsf{o}]
\end{aligned}$$

By $\mathrm{RG}(\Sigma')$ we denote the class of term graphs over Σ' that are correct with respect to some ancestor function. We call $\mathrm{RG}(\Sigma')$ the class of term graphs that *represent* nested term graphs.

Proposition 23. *The transformation I as introduced in Definition 20 gives rise to a well-defined function $I : \mathcal{NG}(\Sigma) \to \mathrm{RG}(\Sigma')$, $\mathcal{N} \mapsto I(\mathcal{N})$ from $\mathcal{NG}(\Sigma)$ into $\mathrm{RG}(\Sigma')$, which preserves $\mathrel{\underline{\rightarrow}}^{\text{ne}}$ as $\mathrel{\underline{\rightarrow}}$, and $\mathrel{\underline{\leftrightarrow}}^{\text{ne}}$ as $\mathrel{\underline{\leftrightarrow}}$.*

This can be proved by keeping the ancestor function of the sntg $\mathcal{G}$ on which the procedure starts, and by extending it appropriately for the vertices in chains of added input vertices below vertices with constant symbols in $\mathcal{G}$. In this way an ancestor function is obtained with respect to which the resulting term graph is correct. Preservation of $\mathrel{\underline{\rightarrow}}^{\text{ne}}$ as $\mathrel{\underline{\rightarrow}}$ along I can be established by arguments using induction on the length of 'access paths' (acyclic paths from the root to a node) in $\mathcal{G}$ and in $I(\mathcal{N})$, respectively.

We note that the image of I is not all of $\mathrm{RG}(\Sigma')$: e.g. the term graph that results from the term graph in Figure 1 right by a homomorphism that identifies all vertices labeled by $\mathsf{i_r}$ is still correct with respect to an ancestor function (compare Lemma 25), but it does not arise as the interpretation of a nested term graph.

As the occurrences of matching output and input vertices in the example of the term graph $I(\mathcal{N})$ in Figure 1 indicate, the nesting structure of a nested term graph is preserved in its term graph interpretation. More importantly, the matching of output and input vertices is guaranteed by the ancestor function of term graphs in $\mathrm{RG}(\Sigma')$. This facilitates the definition of a representation function R from $\mathrm{RG}(\Sigma')$ back to $\mathcal{NG}(\Sigma)$ that is the inverse of I. Similar as for I, also preservation of $\mathrel{\underline{\rightarrow}}$ and $\mathrel{\underline{\leftrightarrow}}$ along R can be shown.

Theorem 24. *Let Σ be an ntg-signature, and let Σ' be defined as in Definition 20. There is a representation function $R : \mathrm{RG}(\Sigma') \to \mathcal{NG}(\Sigma)$ such that $R \circ I = \mathrm{id}_{\mathcal{NG}(\Sigma)}$ holds (that is, R is a retraction of I, and I is a section of R). Furthermore, both of the mappings I and R are efficiently computable. Along I, $\mathrel{\underline{\rightarrow}}^{\text{ne}}$ and $\mathrel{\underline{\leftrightarrow}}^{\text{ne}}$ are preserved as $\mathrel{\underline{\rightarrow}}$ and $\mathrel{\underline{\leftrightarrow}}$, respectively; and along R, $\mathrel{\underline{\rightarrow}}$ and $\mathrel{\underline{\leftrightarrow}}$ are preserved as $\mathrel{\underline{\rightarrow}}^{\text{ne}}$ and $\mathrel{\underline{\leftrightarrow}}^{\text{ne}}$.*

This correspondence opens up the possibility to transfer various well-known results for term graphs to nested term graphs, such as the fact that bisimulation equivalence classes are, modulo isomorphism, complete lattices with respect to homomorphism. Another example is the existence of unique nested term graph collapses, a result whose transfer from $\mathrm{RG}(\Sigma')$ to $\mathcal{NG}(\Sigma)$ depends on the following lemma.

Lemma 25. *The class $\mathrm{RG}(\Sigma')$ of ntgs-representing term graphs is closed under homorphism. That is, if $G_1 \mathrel{\underline{\rightarrow}} G_2$ holds for $G_1, G_2 \in \mathrm{TG}(\Sigma')$, then $G_1 \in \mathrm{RG}(\Sigma')$ implies $G_2 \in \mathrm{RG}(\Sigma')$.*

This proof of this lemma exploits the fact that the term graphs in $\mathrm{RG}(\Sigma')$ are 'fully back-linked' in the following sense: for every vertex w and every output vertex v that in the nesting structure resides above w, there is a path (in forward direction) from w to v. (Since in particular the root output vertex is reachable, this entails that in fact all other vertices are reachable by paths from w.) It follows that if a homomorphism ϕ from an ntg-representing term graph $G_1 \in \mathrm{RG}(\Sigma')$ to a term graph $G_2 \in \mathrm{TG}(\Sigma')$ identifies two vertices w_1 and w_2, then, due to the local progression clauses of the homomorphism and due the ancestor function on G_1, ϕ also identifies all corresponding output vertices in the nesting hierarchy above w_1 and w_2, respectively. This fact makes it possible to define an ancestor function on G_2 for which G_2 is correct. Hence $G_2 \in \mathrm{RG}(\Sigma')$.

Theorem 26. *Every nested term graph $\mathcal{N}$ has, up to isomorphism, a unique nested term graph collapse.*

5 Further aims

We are interested in, and have started to investigate, the following further topics:

Context-free graph grammars We want to view rgs's as context-free graph grammars in order to recognize rgs-generated nested term graphs as context-free graphs. We expect to find a close connection.

Proofnets Formulas containing existential or universal quantifiers, mathematical expressions containing integrals or derivatives, and more generally any language with binding constructs, can be represented as λ-terms over a simply typed signature. Since proofnets refine the latter, it follows that such languages can be represented as proofnets over a signature typed with (MELL) formulas from linear logic. Such a representation should tie in with the boxed representations for first- and higher-order terms of the introduction, but now for nested term graphs. On the one hand, we expect that the development of nested term graphs can profit, via this route, from the detailed studies of the fine structure of proofnets, e.g. various notions of explicit substitution, that have been carried out in the literature (for example, see Accattoli and Guerrini [1]). On the other hand, it is conceivable that the theory of proofnets could benefit from work on nested term graphs for what concerns the natural formalization of infinite nesting, the concepts of bisimilarity and nested bisimilarity, and the faithful representation of nested term graphs as first-order term graphs.

Boxes Extending the previous item, we want to investigate the connection between nested term graphs and the way how boxes that symbolize scopes are employed in various settings. Apart from the box of linear logic proofnets, we are thinking of monads in category theory. These have been introduced and studied to express nested first-order signatures by Lüth [13], and Lüth and Ghani [14], leading to categorical proofs of modularity results in term rewriting.

We would like to obtain a categorical semantics via algebras and coalgebras by viewing nested term graphs as monads over some signature, analogous to how this has been done for first-order term graphs by Ghani, Lüth and de Marchi [6]. Moreover, we would like to understand whether, and if so how, the respective monadic views can be related via our representation of nested term graphs as first-order term graphs. In this respect, the decomposition of boxes into the opening and closing 'brackets' that are used in optimal graph reduction techniques for the λ-calculus, as studied from a categorical perspective by Asperti [2], should be relevant.

Rewrite theory In the above we have only addressed the *static* aspects, how to represent structures with a notion of scope. Ultimately, our interest is in *dynamic* aspects, in rewriting systems for such structures. In particular, since higher-order terms have a natural interpretation as nested term graphs, it is desirable to investigate implementations of higher-order rewriting by nested term graph rewriting, and eventually, via the correspondence explained in Section 4, by first-order term graph rewriting. Several preliminary investigations into this have been carried out, but from different perspectives (corresponding to the different perspectives on boxes as in the previous item): Lafont presents proofnet reduction as reduction on 'nested interaction nets' in [12], Van Raamsdonk defined rewriting modulo proofnets in [17], and Lüth and Ghani defined and studied monadic rewriting in their cited papers. We first want to develop a notion of rewriting on nested terms graphs that is adequate with respect to higher-order term rewriting (HRSs see e.g. [19]), analogous to the adequacy of first-order term graph rewriting for first-order term rewriting [10]. To that end, a notion of equivalence on nested term graphs has to be developed that represents $\alpha\beta\eta$-equivalence on HRS-terms.[7] This also gives rise to other questions, cf. [18], such as how to recognize (efficiently) whether a given (first-order representation of a) nested term graph represents a higher-order term. Next, nested term graph rewriting should facilitate a sensible meta-theory. Here we may think of suitable notions of orthogonality, or termination techniques such as recursive path orders, similar to what has been done for first-order term graph rewriting (see work by Plump [15]).

Acknowledgement We thank the referees for many valuable comments on earlier versions of this paper, and for insisting on referring to related work. We also thank the editors for their patience.

References

[1] B. Accattoli & S. Guerrini (2009): *Jumping Boxes*. In Erich Grädel & Reinhard Kahle, editors: *Computer Science Logic*, *Lecture Notes in Computer Science* 5771, Springer Berlin Heidelberg, pp. 55–70, doi:10.1007/978-3-642-04027-6_7.

[2] A. Asperti (1995): *Linear Logic, Comonads And Optimal Reductions*. *Fundamenta Informaticae* 22(1,2), pp. 3–22, doi:10.3233/FI-1995-22121. Available at `http://dl.acm.org/citation.cfm?id=2383063.2383064`.

[3] R. Bird & R. Paterson (1999): *De Bruijn Notation as a Nested Datatype*. *Journal of Functional Programming* 9(1), pp. 77–91, doi:10.1017/S0956796899003366. Available at `http://www.soi.city.ac.uk/~ross/papers/debruijn.html`.

[4] S. Blom (2001): *Term Graph Rewriting – Syntax and Semantics*. Ph.D. thesis, Vrije Universiteit Amsterdam.

[5] N. Bourbaki (1954): *Éléments de mathématiques: Théories des ensembles*. Hermann.

[6] N. Ghani, C. Lüth & F. de Marchi (2005): *Monads of coalgebras: rational terms and term graphs*. *Mathematical Structures in Computer Science* 15, pp. 433–451, doi:10.1017/S0960129505004743.

[7] C. Grabmayer & J. Rochel (2013): *Term Graph Representations for Cyclic Lambda Terms*. In: *Proceedings of TERMGRAPH 2013*, *EPTCS* 110, pp. 56–73, doi:10.4204/EPTCS.110. Extending report: arXiv:1308.1034.

[8] C. Grabmayer & J. Rochel (2014): *Maximal Sharing in the Lambda Calculus with letrec*. In: *Proceedings of ICFP '14, September 1–6, 2014, Gothenburg, Sweden*, pp. 67–80, doi:10.1145/2628136.2628148.

[9] R.J.M. Hughes (1982): *Supercombinators: A new implementation method for applicative languages*. In: *LFP '82: Proceedings of the 1982 ACM symposium on LISP and functional programming*, pp. 1–10, doi:10.1145/800068.802129.

[10] J. R. Kennaway, J. W. Klop, M. R. Sleep & F. J. de Vries (1994): *On the Adequacy of Graph Rewriting for Simulating Term Rewriting*. *ACM Trans. Program. Lang. Syst.* 16(3), pp. 493–523, doi:10.1145/177492.177577.

[11] E. Kohlbecker, D.P. Friedman, M. Felleisen & B. Duba (1986): *Hygienic Macro Expansion*. In: *Proceedings of the 1986 ACM Conference on LISP and Functional Programming*, LFP '86, ACM, pp. 151–161, doi:10.1145/319838.319859.

[7] Analogous to proofnet reduction. Rewriting should 'interact nicely' with boxes.

[12] Y. Lafont (1990): *Interaction Nets*. In: *POPL '90*, ACM Press, pp. 95–108, doi:10.1145/96709.96718.

[13] C. Lüth (1997): *Categorial Term Rewriting: Monads and Modularity*. Ph.D. thesis, University of Edinburgh.

[14] C. Lüth & N. Ghani (1997): *Monads and modular term rewriting*. In E. Moggi & G. Rosolini, editors: *Category Theory and Computer Science*, *Lecture Notes in Computer Science* 1290, Springer Berlin Heidelberg, pp. 69–86, doi:10.1007/BFb0026982.

[15] D. Plump (1999): *Term Graph Rewriting*. In H. Ehrig, G. Engels, H.-J. Kreowski & G. Rozenberg, editors: *Handbook of Graph Grammars and Computing by Graph Transformation*, 2, World Scientific, pp. 3–61, doi:10.1142/9789812815149_0001.

[16] W. Van Orman Quine (1940): *Mathematical Logic*. Harvard University Press, Cambridge, Mass.

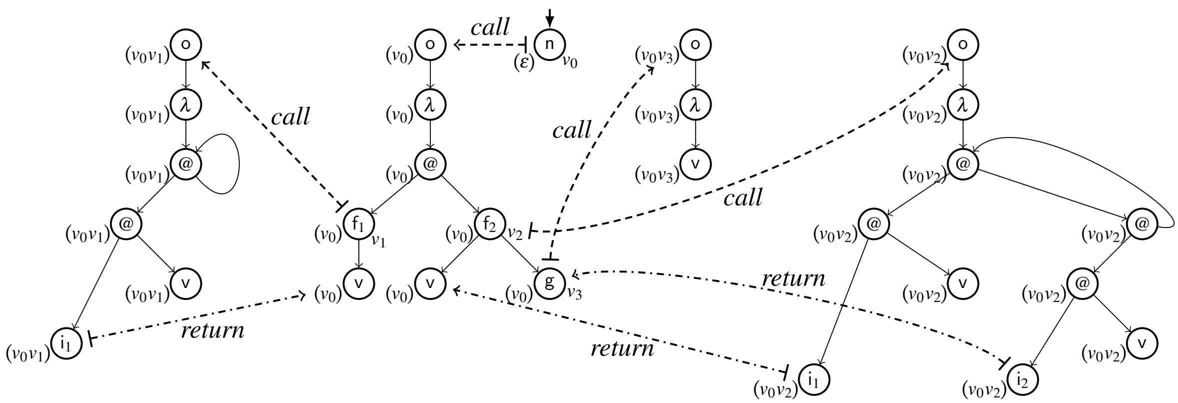
[17] F. van Raamsdonk (1996): *Confluence and Normalisation for Higher-Order Rewriting*. Ph.D. thesis, Vrije Universiteit Amsterdam.

[18] M.R. Sleep, M.J. Plasmeijer & M.C.J.D. van Eekelen (1993): *Term Graph Rewriting – Theory and Practice*. John Wiley & Sons.

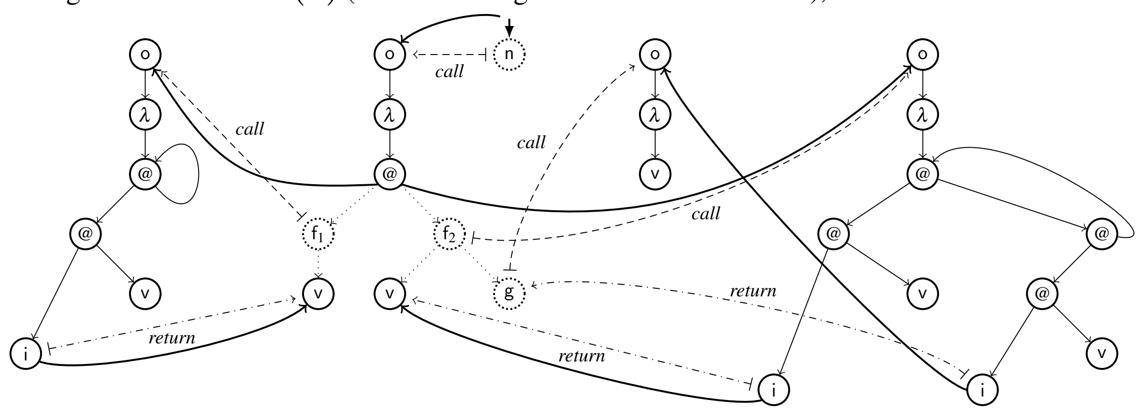
[19] Terese (2003): *Term Rewriting Systems*. *Cambridge Tracts in Theoretical Computer Science* 55, Cambridge University Press.

Appendix A: Interpretation of nested term graphs by first-order term graphs

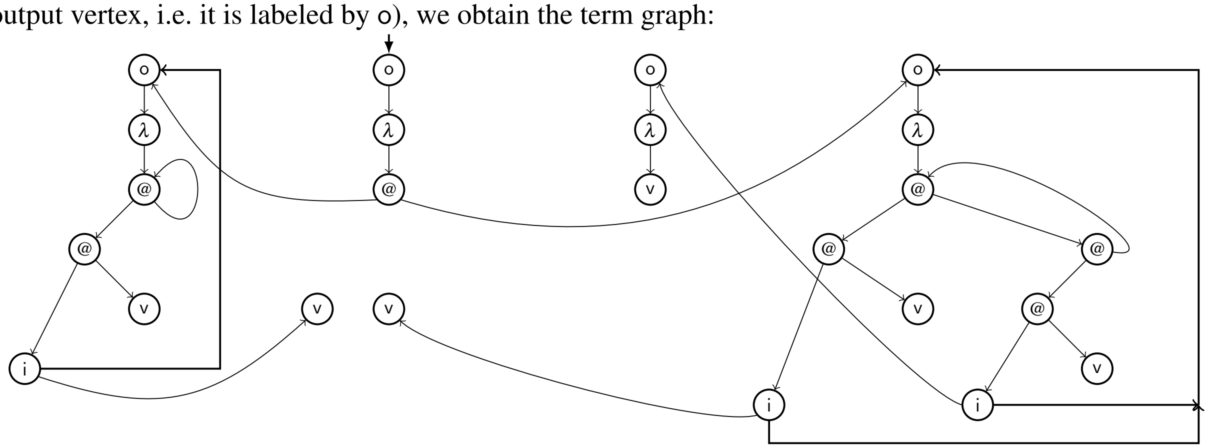
We showcase the transformation process according to Definition 20 of a nested term graph $\mathcal{N}$ into its first-order term graph interpretation $I(\mathcal{N})$ for the example of the nested term graph defined in Example 4 and Figure 3. We start from the sntg-representation $\mathcal{G}$ of $\mathcal{N}$ as illustrated in Figure 5:

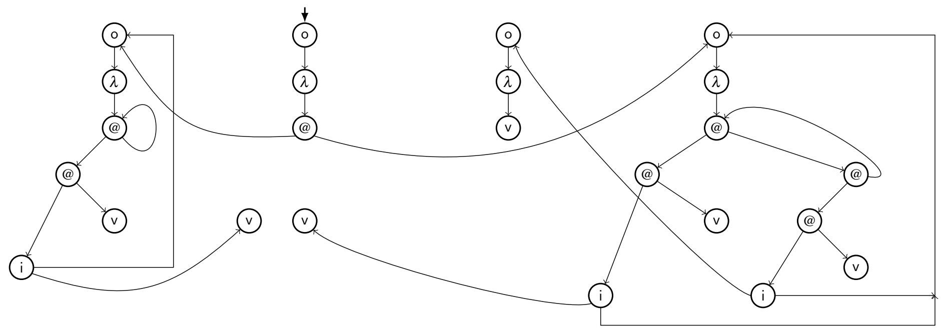


By (i) removing every vertex v with a nested symbol, redirecting incoming edges at v to the vertex $call(v)$, and (ii) relabeling every input vertex w with the nullary label i_k by the binary label i, thereby directing the first edge from w to $return(w)$ (the second edge will be dealt with later), we obtain:

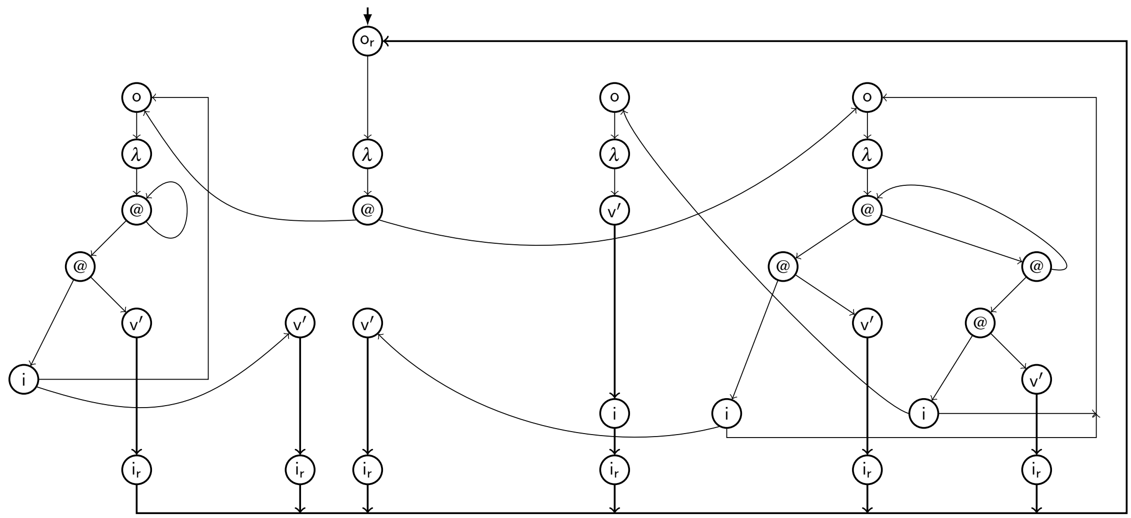


Then by (ii)′ directing the second edge from an input vertex w to become a backlink from w to the corresponding output vertex, that is, the vertex v_n where $anc(w) = v_1 \dots v_n$ (then v_n is guaranteed to be an output vertex, i.e. it is labeled by o), we obtain the term graph:





(The downscaled term graph above repeats the last one from the previous page.) Finally, (iii) by relabeling the output vertex at the root with the special symbol $\mathsf{o_r}$, (iv) by changing the arity of variable vertices (labeled by v) from zero to one (labeled by v'), and letting the outgoing edges target a chain of new binary input vertices through the nesting structure (again the ancestor function can be used for this purpose) towards outermost input vertices that get label $\mathsf{i_r}$ and backlinks to $\mathsf{o_r}$, we obtain:



This term graph is the result of the transformation process of $\mathcal{N}$ into its first-order term graph interpretation $I(\mathcal{N})$. It is isomorphic to the term graph below right, and also right in Figure 1. In order to facilitate a quick structural comparison with the nested term graph $\mathcal{N}$ on which the input $\mathcal{G}$ of this procedure is based, we also show again, below left, the 'pretty print' of the nested term graph $\mathcal{N}$ (from Figure 1 left).