A Lattice of Abstract Graphs

David Clark and Chris Hankin

Dept. of Computing
Imperial College of Science,Technology and Medicine
180 Queen's Gate, LONDON SW7 2BZ, UK

Abstract. This paper concerns the abstract interpretation of Term Graph Rewriting systems. We introduce a new lattice of abstract graphs; this improves over previous work in that the ordering is defined directly on the graph structure rather than via unravelling into trees. We use the lattice in sketching the correctness of Nöcker's *abstract reduction* algorithm; an efficient strictness analysis used in the Concurrent Clean system.

1 Introduction

Since Mycroft's pioneering work in applying abstract interpretation to functional languages [?], much of the work has been theory-intensive: extensions to higher-order functions, polymorphism, finite representations of recursive types and logic-based approaches are examples. Work on practical implementations has lagged behind. Early results have been disappointing; the most successful systems have been restricted to first-order functions. One interesting example, less restricted than many, is the strictness analyser used in the Concurrent Clean system [?]. Concurrent Clean is based on *term graph rewriting* and uses the functional strategy (described below). The strictness analyser is based on *abstract reduction* rather than abstract interpretation.

In this paper, we consider the correctness of abstract reduction. Although the major application of abstract reduction to date has been strictness analysis, the techniques are more widely applicable. In particular, graph rewriting has been proposed as a unified computational model for functional, logic and imperative programming languages. Whilst we are concerned with term graph rewriting here, it is anticipated that our approach will be applicable to more general rewriting systems. This paper continues a programme of work started in [?] and continued in [?, ?]. We start by reviewing some standard definitions for term graph rewriting (which are mainly from [?]).

Definition 1 (Term Graph). A *term graph* over *Symbol*, a set of symbols of given arity including the nullary symbol Ω representing the undefined graph, is a quadruple:

$$(N, lab, succ, r)$$

where N is a set of nodes
$lab : N \to Symbol$ is a labelling function
$succ : N \to N^*$ is the node successor function
r is a distinguished node called the *root*

Rather than represent a rewrite rule by a pair of graphs, it is represented by a triple; this makes sharing between the two sides of the rule explicit.

Definition 2. A term graph *rewrite rule* is a triple:

$$
\begin{array}{ll}
 & (g, n, n') \\
\text{where} & g \text{ is a graph, i.e. a structure } (N, lab, succ) \\
 & n \text{ is a node in } g\text{; the root of the left hand side of the rule} \\
 & n' \text{ is a node in } g\text{; the root of the right hand side}
\end{array}
$$

such that all variable nodes are accessible from the root of the left hand side of the rule.

In the following, let R be a set of rules:

$$R = R_1 \cup R_2 \cup \ldots \cup R_f$$

where each of the R_i defines some operator F_i, i.e. is a set of rules of the form:

$$R_i = \{r_{i,1}, \ldots, r_{i,j_i}\}$$

where $r_{i,k} = (g_{i,k}, n_{i,k}, n'_{i,k})$, $k = 1 \ldots j_i$ and the label on each of the $n_{i,k}$ is F_i. (In the following, we will drop the i index, when the context makes it clear which R_i we are considering). *Constructor symbols*, in contrast to *operators*, are the symbols that do not label the root node of the left hand side of any rule of R (including the δ-rules). We consider any graph with an operator at its root as a redex, so we cannot deal directly with partial rewrite rules, in which such graphs can also be considered as normal forms. However, one can generally complete a partial rewrite system.

The functional strategy depends on the textual order of the rules; the matching algorithm selects the needed redexes 'from top to bottom, and from left to right'. If $g = F_i(\ldots)$ is the graph to be reduced, the strategy first tries to rewrite it by rule $r_{i,1}$, if not, by $r_{i,2}$, and so on. Then the arguments of $F_i(\ldots)$ are reduced from left to right, in order to match some rule $r_{i,k}$.

We will describe the behaviour of the functional strategy with three operations:

$$
\begin{array}{l}
Match : Graph \times Graph \times \wp(Rule) \to \{tt, ff\} \\
Rewrite : Graph \times \wp(Rule) \to Graph \\
Reduce : Graph \times \wp(Rule) \to Graph
\end{array}
$$

The function *Reduce* reduces a graph to root normal form ('rnf'); a graph is in rnf if the symbol at its root is a constructor. *Rewrite* performs a rewrite step, and *Match* implements the matching algorithm. The functions are defined via the following equations (to be read top to bottom):

$$
\begin{array}{lll}
\mathit{Reduce}(g, rs) & = g & \text{if } in\_rnf\, g \\
& = \mathit{Reduce}(\mathit{Rewrite}(g, rs), rs) & \text{otherwise} \\
\\
Match(g, p, rs) & = tt & \text{if } is\_var(p) \\
& = Match(\mathit{Reduce}(g, rs), p) & \text{if } \neg in\_rnf(g) \\
& = ff & \text{if } root(g) \neq root(p) \\
& = MatchArg(args(g), args(p), rs) & \text{otherwise} \\
\\
MatchArg([\,], [\,], rs) & = tt & \\
MatchArg(g : gs, p : ps, rs) & = MatchArg(gs, ps, rs) & \\
& \quad \text{if } Match(g, p, rs) & \\
& = ff & \text{otherwise}
\end{array}
$$

$$
\begin{array}{lll}
\mathit{Rewrite}(g, r : rs) & = \mathit{Rewrite}(g, rs) & \text{if } root(g) \neq root(r) \\
& = \mathit{Rewrite}(g, rs) & \text{if } \neg MatchArg(args(g), args(r), r : rs) \\
& = tgr\, g\, r & \text{otherwise}
\end{array}
$$

where *tgr* is the basic graph rewrite step [?].

Later, we will define abstract versions of these operations; together these will define the notion of "Abstract Reduction" [?]. In order to demonstrate the correctness of the approach, we re-express abstract reduction as an abstract interpretation [?, ?]. The first step in formalising an abstract interpretation is to define a *static* [?] or *collecting* [?] semantics. In general, the role of such a semantics is to collect information about the program's behaviour at certain *program points.* In [?] we have suggested that rules should be used as program points and that the information collected should consist of graphs and pattern matching information which records how the rule has been used. The collecting interpretation is defined once only. The next step is to define the abstract interpretation: we must select a suitable abstract domain of graphs and define abstract operations on elements of the domain. If the abstract domain is related to the collecting interpretation sets by a Galois connection[1], then the appropriate operations may be induced and correctness of the abstract interpretation follows [?]. This approach is good for analyses which concern the intensional behaviour of the program and our previously cited paper contains such an example. In this paper we will be concerned with extensional analyses; as a consequence, we are only concerned with input/output behaviour and correctness can be expressed in

[1] A Galois connection is a pair of monotonic functions (α, γ) such that:

$$\alpha \circ \gamma \sqsubseteq id$$

and

$$\gamma \circ \alpha \sqsupseteq id$$

In abstract interpretation, it is usual to choose the pair of maps such that the first inequality becomes an equality. Such a pair of maps are said to be *exactly adjoined.*

relation to the above operational semantics. We will not consider the collecting semantics further here.

In the next section we introduce some basic concepts relating to contexts and equivalences between graphs. Then, in section 3, we introduce the particular class of abstract graphs that we are working with and an associated partial order which functions as an abstraction order. Section 4 establishes a Galois Connection between abstract graphs and sets of graphs. The final main section presents abstract reduction and considers its correctness and we conclude in section 6.

2 Some Basic Concepts

Definition 3 (Many-Holed Context). A many-holed context is a term graph with zero or more nodes labelled []. Each node so labelled has no successors. We use the following terminology when discussing many-holed contexts:

Let $C[]^k$ represent a graph context with k holes, if $k = 0$ then $C[]^k$ has no holes, otherwise the holes are labelled 1 to k.

Alternatively, $C[]^k$ is represented as $C[]_1 \dots []_k$.

What sort of graph can be put in a hole in a context? (From now on we will use the word context to refer to a many-holed context unless we explicitly state otherwise). Given that the graphs represent terms, we want a graph corresponding to a term to be able to fill the hole. Once the hole is filled, the graph that filled the hole must in some sense be a sub graph of the graph produced by filling all the holes. [**?**] defines a notion of a sub graph rooted at a given node as the restriction of the labelling and successor functions of the main graph to the nodes reachable by paths from the root of the sub graph.

Definition 4.
Let $G = (N, lab, succ, r)$ be a term graph, the sub graph of G rooted at n, for $n \in N$, written $G \mid n$ is:

$$(N', lab', succ', n)$$

where $N' = \{n' \in N \mid \text{there is a path from } n \text{ to } n'\}$ and lab' and $succ'$ are the restrictions of lab and $succ$ to N'.

This definition is not quite narrow enough as the root of the sub graph must be the only node at which the sub graph is connected to the main graph if it is to be characterised as filling a hole in a context. Before we can give a suitable definition of sub graph it is useful to make some supporting definitions including the following which is marginally different form [**?**].

Definition 5.
$f : G \to H$, where $G, H \in \text{Graph}$,
is homomorphic at a node n in G iff

$$
\begin{aligned}
lab_H(f(n)) &= lab_G(n) \text{ and} \\
f(succ_G(n)_i) &= succ_H(f(n))_i, \text{ for every } i \\
&\quad \text{where } succ(n)_i \text{ is the } i^{\text{th}} \text{ successor of } n
\end{aligned}
$$

So a function between graphs is homomorphic at a node if it preserves the label at that node as well as the successors of the node and their order. A homomorphism at each node between a graph and part of another graph captures some of the idea of a sub graph in the sense that it establishes a close correspondence between the one graph and part of the other.

Definition 6. Given two abstract graphs, G and H, and a mapping $f : G \to H$, f is a total homomorphism if it is homomorphic at each node of G.

From now on the word homomorphism will be used to refer to a total homomorphism. But a homomorphism is not enough as we will require the sub graph to be exactly the same as the part of the larger graph and a homomorphism will allow the target or image graph to have more sharing of nodes than the source graph. (See [?] for an example of this). Thus it will be necessary to have a total homomorphism in each direction - an isomorphism.

Definition 7 (Graph Equivalence, $\approx$).
A homomorphism, f, is *rooted* if $f : G \to H$ and $f(r_G) = r_H$
An isomorphism is a homomorphism with an inverse. Two graphs, G and H, are said to be *equivalent* if there is a rooted isomorphism between them. In which case we write $G \approx H$.

It is now possible to state precisely what is meant by a sub graph in the sense that we intend the concept to be understood. G is a sub graph of H if G is equivalent to a part of H which is a self-contained term graph connected to the rest of H only through its root.

Formally, a sub graph is a graph which fulfils the following two conditions. Firstly it is a sub graph in the sense of Definition **??** and secondly, for any of its nodes except its root all the predecessors of the node are also nodes of the sub graph.

Definition 8.
G is a *rooted term sub graph* of H(henceforth: sub graph), and we write $G \leq_t H$ iff
(i) $\exists n \in N_H$ such that $H \mid n \approx G$ and
(ii) $\forall n' \in N_{H|n}, n' \neq n$, if $\exists n'', i > 0$ such that $n' = succ(n'')_i$ then $n'' \in N_{H|n}$.

The collection of contexts associated with a particular graph, provided with a suitable ordering, forms an upper semi-lattice with the graph as the bottom element and [] as the top element. This result is used in the detailed proofs of the next two sections. We will not pursue this further here, but the details will appear in a full version of the paper.

3 Abstract Graphs

We write the set of symbols with which abstract graphs (A-Graph) are labelled as Symbol$^{\#}$. This is defined as:

Symbol$^{\#}$ $= \{F^{\#} \mid F \in$Symbol, $F \neq \Omega\} \cup \{$ Bot, Top, Union$\}$
where Bot and Top are of arity 0 and Union is of arity 2.

The symbols *Bot* and *Top* represent the 0 and 1 of traditional strictness analysis [?]. Abstract graphs represent sets of concrete graphs; we expect that a graph with *Union* at its root abstracts both of it's immediate sub graphs and hence is analogous to the set union of the singleton sets of those graphs. Graphs with nodes labelled with *Union* are introduced during abstract reduction; a particular abstract graph may "match" many of the rules - the result of reducing the graph is the union of the results from each of the matching rules.

An abstraction ordering on abstract graphs must satisfy constraints such as a graph consisting of a node labelled *Bot* is less than any graph; and any graph is less than a graph consisting of a node labelled *Top*.

We begin by defining a very rudimentary approximation ordering which does not take Unions into account. We want the abstract graph (symbol) Bot to approximate any A-graph, any A-graph to approximate Top, and any A-graph to approximate itself.

Definition 9.
$\forall G \in A-$Graph,
Bot $\sqsubseteq_{\mathrm{bt}} G \sqsubseteq_{\mathrm{bt}}$ Top, and
$G \sqsubseteq_{\mathrm{bt}} G$.

Fact: $\sqsubseteq_{\mathrm{bt}}$ is a Partial Order on A-Graph

Our eventual objective is to use this ordering to define a partial order on abstract graphs which gives a proper account of Union nodes. We achieve this in two further steps. We consider that one abstract graph approximates another if each can be factored into the same context but with different sub graphs filling each hole in the context and each pair of sub graphs associated with a given hole satisfying the above ordering.

Definition 10.
Let $G, H \in A-$Graph then

$$G \sqsubseteq_{\mathrm{ap}} H$$

if there is a many holed context, $C[]^k$, and sets of graphs, $\{g_i\}$ and $\{h_i\}$, such that:
(i) $\forall i, 1 \leq i \leq k, g_i \leq_t G$ and $h_i \leq_t H$.
(ii) $G = C[g_1]_1 \ldots [g_k]_k$ and $H = C[h_1]_1 \ldots [h_k]_k$.
(iii) $\forall i, 1 \leq i \leq k, g_i \sqsubseteq_{\mathrm{bt}} h_i$.

Proposition 11. $\sqsubseteq_{ap}$ *is a Partial Order.*

Now we want to extend this ordering to handle the arbitrary addition of nodes labelled *Union* to abstract graphs. The extension will not be a partial order but a pre-order which is expected since, intuitively, we are trying to model unions of singleton sets of graphs. Thus $Union(A, B)$ must approximate $Union(B, A)$ and vice versa although these are not identical (equivalent) abstract graphs as the order of the successors to the root node is different in each case. In addition we want to define our ordering so that *Bot* models the empty set (i.e. is a unit for union) and *Top* models Graph, the set of all concrete graphs (i.e. is a zero for union). Intuitively, $G \leq_G H$ if, once all the *Union* nodes are stripped out (leaving a collection of sub graphs of each graph), every sub graph of the lesser graph has a sub graph of the other graph which is greater than it with respect to $\sqsubseteq_{\mathrm{ap}}$. In addition, it will be useful to think of the *Union* nodes as being "above" or closer to the root of the graph; this is a simplifying restriction which corresponds closely to the idea that the abstract graphs with *Union* nodes are modelling set union of singleton sets of graphs.

In order to capture the idea of stripping all the *Union* nodes out we need to formally define the largest possible context for a given graph such that the context contains only *Union* labelled nodes. Since we are trying to model set union of singleton sets we would like to restrict our ordering to graphs which either have no *Union*-labelled nodes (so that the new ordering extends the one defined above) or which have a *Union*-labelled root and any other *Union*-labelled node in the graph is a successor of a *Union*-labelled node. Unfortunately, this intuitive account is not correct if the *Union* nodes appear in cycles which also contain non-*Union* nodes; such graphs are incomparable with other graphs except *Top*.

Definition 12. A graph, G, is said to be *Union-rooted* if either
(i) G contains no nodes labelled *Union*
(ii) if $lab_G(n) = Union, n \neq r_G, \exists i > 0$ and $m \in N_G$ such that $lab_G(m) = Union$ and $n = succ_G(m)_i$.

Definition 13. Given a graph, $G \in A-$Graph, the *maximum Union context (mUc) of* G is a many-holed context, $C[]^k$, such that:
(i) $\exists g_1, \ldots g_k \leq_t G$ such that $C[g_1]_1 \ldots [g_k]_k = G$
(ii) all the nodes in $C[]^k$ are labelled *Union* i.e.
$\forall m \in C[]^k, lab_{C[]^k}(m) = Union$
(iii) adding another node from G to $C[]^k$ would violate one of the above properties

We will also need the dual of the above definition - the *maximum Union free context*. The definition is the same except that none of the nodes of the context are labelled *Union* and adding another node from the graph would violate one of the two properties. The intention is that the maximum *Union* free context for a graph will have holes which are filled with *Union*-rooted sub graphs (or root-cyclic[2] sub-graphs).

Since a graph is a context without any holes, the maximum *Union* free context for a graph without any *Union* nodes is the graph itself and the maximum

[2] A *root-cyclic* graph is a cyclic graph which contains a cycle which includes the root.

union context for the same graph is the maximum context, i.e. []. [] is obviously also the maximum *Union* free context for a *Union*-rooted graph.

These definitions can be extended to define functions on $A - Graph$; mUc (resp. $mUFc$) associates each graph with its maximum Union context (resp. maximum Union free context).

We can now define the pre-order which is designed to cope with the arbitrary introduction of *Union* nodes into a graph. The idea is that one abstract graph is less than another if, firstly, they have the same maximum *Union* free context, and, secondly, if g_i fills a hole of the lesser graph and the set of the sub graphs that fill the holes of the maximum *Union* context of g_i is $\{{}^i g_j \mid j$ is the arity of the mUc of $g_i\}$, then for any graph produced by arbitrarily replacing all of the g_i by one of the ${}^i g_j$, there is a corresponding graph produced by the same process applied to the greater graph so that: if both resulting graphs are *Union* free or the lesser is root-cyclic, the graphs satisfy $\sqsubseteq_{\text{ap}}$. Otherwise the process must be done again.

While this may appear unnecessarily complex to the reader, we believe the complexity is necessary for the following reasons: the definition must perforce be inductive since the ordering we are attempting to define must be able to deal with "nested" maximum *Union* contexts; unfortunately, experience has taught us that, to successfully define the Galois connection between Graph and A-Graph it is a wise idea <u>not</u> to throw away the context(s) as one moves through the nested maximum *Union* contexts (i.e. the mUc!!).

Before we can formalise the definition of this pre-order, we need some auxiliary notions. We start by defining a function from $A - Graph$ to lists of $A - Graph$s which produces the list of sub graphs in the order in which they slot into the holes of either the mUc or the mUFc (depending on whether the argument graph is union-rooted or not).

Definition 14 **(**$Unbundle : A - Graph \rightarrow A - Graph^*$**).**
If G is union-rooted then:

$$Unbundle(G) = g_1 \ldots g_n \; where \; G = C[g_1]_1 \ldots C[g_n]_n \; and \; mUc(G) = C[]^n$$

else:

$$Unbundle(G) = g_1 \ldots g_n \; where \; G = C[g_1]_1 \ldots C[g_n]_n \; and \; mUFc(G) = C[]^n$$

Using $Unbundle$, we can define a function $repClass$ as follows:

Definition 15 **(**$repClass : A - Graph \rightarrow \wp(A - Graph)$**).**

$$\begin{aligned} repClass(G) = \{ & C[{}^1 g_{i1}]_1 \ldots [{}^n g_{in}]_n \mid mUFc(G) = C[]^n, \\ & \forall 1 \leq k \leq n. {}^k g \; in \; Unbundle(G) \; and \; {}^k g_{ik} \; in \; Unbundle({}^k g)\} \end{aligned}$$

We can now define the ordering:

Definition 16.
Let $G, H \in A - Graph$, then $G \leq_G H$ iff:
$\forall g \in repClass(G). \exists h \in repClass(H).$
((g and h are both union free or g is root-cyclic) and $g \sqsubseteq_{ap} h$) or $g \leq_G h$

Proposition 17. $\leq_G$ *is a Pre-Order on A-Graph.*

There remains the problem that this is only a pre-order and we need a partial order. We induce a partial order from the pre-order by taking equivalence classes.

Definition 18. Let $G, H \in A-$Graph, $G \equiv_G H$ iff $G \leq_G H$ and $H \leq_G G$.
Let A-Graph$_{/\equiv_G}$ be the set of equivalence classes of elements of A-Graph under $\equiv_G$.
Let $\sqsubseteq_G$ be the partial order on AGraph$_{/\equiv_G}$ induced from $\leq_G$.

Finally, we have defined a partial order, $\sqsubseteq_G$, on A-Graph - well not on A-Graph exactly, but on a set of equivalence classes of A-Graph. Confusingly, we will assume that "A-Graph" refers to A-Graph$_{/\equiv_G}$ from now on unless it is explicitly indicated otherwise. Also confusing may be the use of [G] to represent a member of A-Graph rather than the maximum context with the graph G put in the hole. We will try to make the intended meaning clear from the context!

What do the equivalence classes "look like"? The answer to this is almost as complicated as the definition that produced the classes. The simplest way of answering this question is to prove some properties of the classes, some of which are desirable ones mentioned earlier; the reader is invited to compare these results with [**?**] where we employed considerable extra machinery. Here are some equivalences that "fall" out of the definition of $\leq_G$.

Proposition 19. $\forall A, B, C, F[] \in A-$*Graph, where* $F[]$ *is a context with one hole,*

$$\begin{array}{ll}
\textit{[Union(A,B)]} & \equiv_G \textit{[Union(B,A)]} \\
\textit{[Union(Union(A,B),C)]} & \equiv_G \textit{[Union(A,Union(B,C))]} \\
\textit{[Union(A,A)]} & \equiv_G \textit{[A]} \\
\textit{[Union(A,Bot)]} & \equiv_G \textit{[A]} \\
\textit{[Union(A,Top)]} & \equiv_G \textit{[Top]} \\
\textit{[F[Union(A,B)]]} & \equiv_G \textit{[Union(F[A],F[B])]}
\end{array}$$

Proof. The maximum Union free context for Union(A,B) and Union(B,A) is [].
The set of "sub sub graphs" in each case is $\{A, B\}$ and since $\leq_G$ is reflexive we have Union(A,B) $\leq_G$ Union(B,A) and Union(B,A) $\leq_G$ Union(A,B).

The other proofs are very similar and we shall now only indicate the proof of the most interesting, last identity.
The maximum Union free contexts for F[Union(A,B)] and Union(F[A], F[B]) are F[] and [] and the "sub sub graphs" are $\{A, B\}$ and $\{F[A], F[B]\}$ respectively.
The rest is straightforward.

□

It is now possible to show that A-Graph is an upper semi-lattice and the least upper bound of any two classes, [G] and [H], is given by [$Union$(G, H)].

Proposition 20. *A-Graph is an Upper Semi-Lattice.*

Proof. We prove that [G] $\vee$ [H] = [Union(G, H)]:
First, we prove that [G] $\sqsubseteq_G$ [Union(G,H)], [H] $\sqsubseteq_G$ [Union(G,H)]:

Choose Union$(G, G) \in [G]$ and Union$(G, H) \in$ [Union(G, H)]. The maximum Union free contexts on Union(G, G) and Union(G, H) are both [] and the "sub sub graphs" are $\{G\}$ and $\{G, H\}$respectively. Since $\leq_G$ is reflexive we have:

Union(G, G) $\leq_G$ Union(G, H) and hence [G] $\sqsubseteq_G$ [Union(G, H)]. Similarly for H.
Now suppose [G] $\sqsubseteq_G$[K] and [H] $\sqsubseteq_G$ [K]. [G] $\sqsubseteq_G$ [K] $\Rightarrow$ G $\leq_G$ K and [H] $\sqsubseteq_G$ [K] $\Rightarrow$ H $\leq_G$ K. It is then straightforward to show that Union(G,H) $\leq_G$ Union(K,K) and hence that [Union(G, H)] $\sqsubseteq_G$ [K].

□

We conjecture that this result can be extended to show that (A-Graph, $\sqsubseteq_G$) is a complete lattice with [Top] as the greatest element and [Bot] as the least element.

4 The Galois Connection

In order to demonstrate the correctness of abstract reduction we define an exactly adjoined pair of maps between the lattice of sets of concrete graphs ordered by set inclusion and the lattice of abstract graphs ordered by $\sqsubseteq_G$.

We define a monotonic and onto abstraction map and then induce the concretisation map in such a way that the pair are exactly adjoined.

We begin by defining a map from concrete graphs to the equivalence classes of abstract graphs. We think of this map as the "inclusion" map because it maps a single concrete graph into the class of abstract graphs that includes that concrete graph.

Definition 21 (inc: Graph $\rightarrow A-$Graph).
Let $G \in$ Graph, with $G = (N, lab, succ, r)$, then $G' = (N, lab', succ, r)$, where $\forall n \in N$:

$$\begin{aligned} lab'(n) &= A^{\#}, \text{ if } lab(n) = A \text{ and } A \neq \Omega \\ &= \text{Bot, if } lab(n) = \Omega \end{aligned}$$

We define
$inc(G) = [G']$

We then use this map to define the abstraction map as mapping a set of concrete graphs to the least upper bound of the set of equivalence classes of abstract graphs produced by applying the inclusion map to each member of the set of concrete graphs in turn.

Definition 22 ($\alpha : \wp($Graph$) \rightarrow A-$Graph).
$\forall S \in \wp(\text{Graph}),\ \alpha(S) = \bigsqcup\{\text{inc}(G) \mid G \in S\}$

Proposition 23. *α is monotonic and onto.*

Proof.
α is monotonic:

Let $S, T \in \wp(\text{Graph})$ and $S \subseteq T$. Consider $\alpha(S)$ and $\alpha(T)$. By the definition of α, these both have representatives, A and B respectively, which are Union-rooted graphs with no nesting of maximum Union contexts. The maximum Union free context for each representative is [] and the "sub sub graphs" in each maximum Union context consist of inc applied to the members of S and T respectively. Since $S \subseteq T$ and $\sqsubseteq_{\text{ap}}$ is reflexive, we have A $\leq_G B$ and hence $\alpha(S) \sqsubseteq_G \alpha(T)$.

α is onto:

Let $[G] \in A{-}\text{Graph}$. Since
$\forall A, B, F[] \in$ A-Graph, where $F[]$ is a context with one hole,

$$[F[\text{Union}(A, B)]] \equiv_G [\text{Union}(F[A], F[B])]$$

we have that any member of A-Graph has a representative which is a union rooted graph with no nesting of maximum Union contexts. If the *lab* map associated with inc was onto Symbol$^{\#}$, this would be sufficient, since every Union-rooted graph with no nesting of maximum Union contexts is then the image of a set of concrete graphs. But *lab* will never label any node as Top. However Top is bigger than every member of A-Graph with respect to any of the orderings defined. So $\alpha(\wp(\text{Graph})) = [\text{Top}]$. Thus if G is F[Top] where F[] is a BUT-free, one holed context, then

$[G] = \alpha(\{F[H] \mid H \in \text{Graph }\})$

$\square$

The concretisation map can now be induced from α.

Definition 24 ($\gamma : A{-}$Graph $\rightarrow \wp($Graph$)$).
$\gamma(G) = \bigsqcup\{H \mid \alpha(H) \sqsubseteq_G [G]\}$

5 Abstract Reduction

The abstract reduction process is achieved by three main functions, which parallel those above, plus a number of auxiliary functions. The main functions are:

$$\begin{array}{l} AReduce : A - Graph \times \wp(ARule) \rightarrow A - Graph \\ ARewrite : A - Graph \times \wp(ARule) \rightarrow A - Graph \\ AMatch : A - Graph \times A - Graph \times \wp(ARule) \rightarrow Matching \end{array}$$

The definition of *AReduce* is analogous to that of *Reduce*:

$$\begin{array}{rll} AReduce(g, rs) & = g & \text{if } in - rnf(g) \\ & = AReduce(ARewrite(g, rs), rs) & \text{otherwise} \end{array}$$

However, the other functions are considerably more complicated. The process of abstraction throws information away; as a consequence an abstract graph may match several rules where the concrete graph would have only matched one (according to the functional strategy). Moreover, the abstract graph may "match" the left hand side of a rule in different ways; the set of possible matchings, $Matching$, has four elements:

TotalMatch : Every graph in the concretisation of the abstract graph matches the pattern.

NoMatch : No graph in the concretisation of the abstract graph matches the pattern.

BotMatch : The matching of any graph in the concretisation of the abstract graph against the pattern leads to divergence.

PartMatch : Some of the concrete graphs match the pattern

$ARewrite$ is now defined as follows:

$$
\begin{aligned}
ARewrite(Fv_1 \ldots v_n, []) &= Bot \\
ARewrite(Fv_1 \ldots v_n, r : R) &= tgr(Fv_1 \ldots v_n, r), \\
&\quad \text{if } AMatchAlt(Fv_1 \ldots v_n, p) = TotalMatch \\
&= ARewrite((Fv_1 \ldots v_n, R), \\
&\quad \text{if } AMatchAlt(Fv_1 \ldots v_n, p) = NoMatch \\
&\quad \vee\, root(p) \neq F \\
&= Union(tgr(Fv_1 \ldots v_n, r), \\
&\quad ARewrite(Fv_1 \ldots v_n, R)), \\
&\quad \text{if } AMatchAlt(Fv_1 \ldots v_n, p) = PartMatch \\
&= Bot, \text{ otherwise}
\end{aligned}
$$

where

$$
\begin{aligned}
p &= \text{"lhs of r"} \\
AMatchAlt(g, p) &= AMatchArgs(args(g), args(p), r : R), \\
&\quad \text{if } root(g) = root(p) \\
&= NoMatch, \text{ otherwise}
\end{aligned}
$$

Finally, we define $AMatch(Args)$:

$$
\begin{aligned}
&AMatchArgs([], [], rs) = nil \\
&AMatchArgs((g : gs), (p : ps), rs) = \\
&CombineMatch(AMatch(g_1, p_1, rs) : AMatchArgs(gs, ps, rs))
\end{aligned}
$$

where

$$
\begin{array}{ll}
AMatch(g,p,rs) & = TotalMatch, \text{ if } is-var(p)\\
AMatch(g,p,rs) & = AMatch(AReduce(g,rs),p,rs), \text{ if } \neg in-rnf(g)\\
AMatch(Top,p,rs) & = PartMatch\\
AMatch(Bot,p,rs) & = BotMatch\\
AMatch(g,p,rs) & = CombineUMatch(AMatch(g_1,p,rs),AMatch(g_2,p,rs)),\\
 & \quad \text{if } root(g) = Union\\
AMatch(g,p,rs) & = NoMatch, \text{ if } root(g) \neq root(p)\\
AMatch(g,p,rs) & = AMatchArgs(args(g),args(p),rs), \text{ otherwise}
\end{array}
$$

and

$$
\begin{array}{ll}
CombineMatch(m_1,\ldots,m_n) & = TotalMatch,\\
 & \quad \text{if } m_i = TotalMatch \text{ for all } i\\
 & = NoMatch,\\
 & \quad \text{if } m_i = NoMatch \text{ for some } i \wedge\\
 & \quad m_j = TotalMatch \text{ for all } j < i\\
 & = BotMatch,\\
 & \quad \text{if } m_i = BotMatch \text{ for some } i \wedge\\
 & \quad m_j = TotalMatch \text{ for all } j < i\\
 & = PartMatch, \text{ otherwise}
\end{array}
$$

$$
\begin{array}{ll}
CombineUMatch(m_1,m_2) & = TotalMatch,\\
 & \quad \text{if } m_i = TotalMatch \text{ for all } i\\
 & = NoMatch,\\
 & \quad \text{if } m_i = NoMatch \text{ for all } i\\
 & = BotMatch,\\
 & \quad \text{if } m_i = BotMatch \text{ for all } i\\
 & = PartMatch, \text{ otherwise}
\end{array}
$$

Theorem 25 (Correctness).

$$\forall g \in Graph.\forall R \in \wp(Rule).Reduce(g,R) \in \gamma(AReduce(inc(g),R^{\#}))$$

The proof of which uses the following lemma:

Lemma 26.
$\forall R \in \wp(Rule)$

$$
\begin{array}{lll}
\textit{(i)} & AMatch(inc(g),inc(g'),R^{\#}) = NoMatch & \Rightarrow \neg Match(g,g',R)\\
\textit{(ii)} & AMatch(inc(g),inc(g'),R^{\#}) = BotMatch & \Rightarrow Match(g,g',R)\\
 & & \quad \textit{diverges}\\
\textit{(iii)} & AMatch(inc(g),inc(g'),R^{\#}) = TotalMatch & \Rightarrow Match(g,g',R)\\
\textit{(iv)} & AMatch(inc(g),inc(g'),R^{\#}) = & \\
 & TotalMatch \vee PartMatch & \Leftarrow Match(g,g',R)
\end{array}
$$

Proof (Correctness Theorem). The theorem is proved in conjunction with the above lemma by a straightforward (but tedious) inductive argument. □

6 Conclusions

We have developed a new ordering for abstract graphs. This improves on [**?**] in that:

- It is defined directly in terms of graphs rather than via unravellings into infinite trees. As a consequence, sharing is preserved directly rather than by the cumbersome annotations introduced in [**?**].
- Many properties, for example those in Proposition **??**, have elegant proofs in this new ordering. We expect this to carry over to the detailed proofs of the properties of abstract reduction.

Acknowledgements

The second author was partially funded by Esprit Working Group 6345 (Sema-Graph II). The work reported here would not have been possible without our previous collaboration with Eric Goubault, Eric Nöcker and Marko van Eekelen. It was John Hughes who first pointed out that it might be better to define an ordering on graphs directly rather than via unravellings into trees.