Safety of Strictness Analysis via Term Graph Rewriting

David Clark[1], Chris Hankin[1], and Sebastian Hunt[2]

[1] Department of Computing, Imperial College, London SW7 2BZ
{djc,clh}@doc.ic.ac.uk
[2] Department of Computing, City University, London EC1V 0HB
seb@soi.city.ac.uk

Abstract. A safe abstraction is presented for a restricted form of term graph rewriting. This abstraction can be seen as a formalisation of the rewrite system employed by the strictness analyser in the Concurrent Clean compiler. Programs written in a core functional language are interpreted as graph rewriting systems using a form of equational term graph rewriting due to Ariola and Arvind. Abstract graphs are defined by extending the signature of ordinary graphs and it is shown how to extend a rewriting system on ordinary graphs to one on abstract graphs. An abstraction relation between abstract graphs is used to define a notion of safety with respect to a variant of Ariola and Arvind's direct approximation semantics, and this notion of safety is shown to be adequate for strictness analysis. Abstract reduction is defined as the union of the extended rewrite system with additional 'heuristic' reductions and shown to be safe.

1 Introduction

In this paper we present a safe abstraction for a restricted form of term graph rewriting system and show how it can form the basis of a strictness analyser for an untyped core functional language. This work grows out of work by Eric Nöcker on the strictness analyser used in the Clean compiler [16, 17] and subsequent attempts to provide a satisfactory formal account of the method.

1.1 Background

Nöcker's 1988 paper on the strictness analyser used in the Clean compiler [16, 17] did not give a formal account of the algorithm. Claims made for the algorithm included the ability to find higher order strictness information, a high speed of execution of the algorithm and a high speed of execution of the resulting code. (Apart from some material in Nöcker's thesis [18] little has been published to back up these claims.) As a method of strictness analysis it seems formidable in practice but the lack of a formalisation meant that it was not possible to prove that the algorithm was correct. Surprisingly, this has remained the situation until the time of writing.

The algorithm can be seen as having two parts: a method of rewriting so-called abstract graphs together with a collection of heuristics which are used to safely force termination of rewrite sequences. In this paper we provide the first complete account of the first part of the algorithm and account for some, but not all, of the heuristics.

The Clean strictness analyser works by manipulating abstract graphs, which are just graphs over an extended signature. The additional symbols in the extended signature are used to allow a single abstract graph to represent a set of ordinary graphs. In this, Nöcker's work is clearly inspired by abstract interpretation [10, 1, 15]. The intuitions behind the symbols used to construct abstract graphs are as follows: $\bot$ represents Ω, the undefined graph; $\top$ represents all graphs; union symbols (rendered as $\oplus$ below) are used to represent a point of non-deterministic choice within an abstract graph. (In the formalisation of the current paper, $\bot$ is indistinguishable from Ω and so the only additional symbols used are $\top$ and $\oplus$.)

1.2 Related Work

There are a number of earlier papers which attempt to formalise the algorithm. We briefly review these.

The first and (hitherto) most complete formal account of the strictness analyser, is found in a pair of papers by Goubault et al [12, 22], which show how to place the algorithm into an abstract interpretation framework. The first of these papers [12] uses the syntax of abstract graphs to define a class of the lattices required by abstract interpretation: each abstract graph is unravelled to a set of regular trees and a standard ordering on trees is used to induce an ordering on abstract graphs. There are two drawbacks with this approach. Firstly, sharing information is lost. Secondly, to have a lattice based on regular trees, graphs containing cycles via $\oplus$ have to be disallowed and precisely such graphs are used in a key way in the Clean strictness analyser to represent lists. The second paper [22] shows the rewriting of abstract graphs and the heuristics to be safe for this restricted class of abstract graphs.

Nöcker, a co-author of [22], made another attempt to formalise the algorithm in a paper [18] which gives a formal framework for abstract reduction for term rewriting. The paper speculates on extending the formalisation to term graphs via infinitary term rewriting but falls short of completing the extension. Hankin and Clark [8] attempted to improve upon the the ordering in [12] but their ordering also excludes cycles via $\oplus$.

The approach taken by Schütz et al [20] is, in effect, to do away with union symbols altogether and develop a theory based on non-deterministic rewriting. (The paper does contain a brief informal description of how unions are used to obtain an efficient implementation. No attempt is made to formalise this use of unions or prove it correct.) The result is interesting but it uses term rewriting rather than graph rewriting and so does not apply directly to the Clean strictness analyser. However this work inspired a related approach to term graph rewriting by Clark [9] which sketched a proof for the correctness of abstract reduction.

Although our formalisation is *not* presented as an abstract interpretation, there are striking similarities (see Sect. 7). Despite these similarities, it is not yet clear whether our approach can be alternatively formalised as an abstract interpretation.

One distinctive feature of our work (and Nöcker's original work) is that while $\oplus$ seems to play the role taken by $\sqcup$ in abstract interpretation, it is treated in a purely syntactic manner, not as the join operation in a lattice. In this, our work is reminiscent of Chuang and Goldberg's syntactic approach to fixed point computation [7]. More generally, the use of direct syntactic/operational approaches in reasoning about program behaviour has proved very successful in recent years (see [11] for some good examples). An example which is close in spirit to the style of reasoning employed in the current paper is the use of the notion of subject-reduction in the work of Wright and Felleisen [25].

1.3 Our Contribution

In the current paper we avoid the limitations of earlier approaches by working directly with graphs, defining an abstraction relation between abstract graphs, and defining a weak notion of homomorphism which allows us to compare abstract graphs directly without unravelling. Our definitions do not give us a lattice of abstract graphs. Nonetheless, as we show, using these definitions it is possible to state and prove safety in a direct operational manner.

We believe that our formalisation is complete with respect to the clean strictness analyser, in the following sense: all those abstract graphs which are used in the analyser are allowed by our framework and we conjecture that no alternative to our notion of safety will allow more strictness properties to be inferred from the results of abstract reduction. However, our work does not constitute a complete proof of correctness for the analyser since it does not address the 'loop detection' heuristics which are used to force termination (see Sect. 6 and Sect. 7).

1.4 Outline

In Sect. 2 we define a functional language and interpret its programs as term graph rewriting systems of a restricted form (which we call simple functional GRSs). In Sect. 3 we define abstract graphs and an abstraction relation between ordinary and abstract graphs, and show how a simple functional GRS on ordinary graphs can be extended to one on abstract graphs. In Sect. 4 we define a variant of a direct approximation style semantics for our restricted class of rewriting systems. In Sect. 5 we state our notion of safety using this semantics and show it to be adequate for strictness analysis. In Sect. 6 we define abstract reduction and show how to prove it safe by establishing that it commutes with the extended simple functional GRS introduced in Sect. 3. Finally, in Sect. 7 we summarise and suggest directions for future work. Proofs of key results are sketched in an appendix.

2 Graph Rewriting Systems

Following Barendsen and Smetsers [6], we describe a core functional language and show how its programs can be interpreted as term graph rewriting systems.

Let x, y, z range over a set Var of variables. A set $\mathcal{F}$ of function symbols and a set $\mathcal{C}$ of constructor symbols, determines a signature $\mathcal{S} = \mathcal{F} \cup \mathcal{C}$. All symbols in $\mathcal{S}$ are assumed to have a fixed arity: let $\mathcal{S}^k$ ($\mathcal{F}^k$, $\mathcal{C}^k$) denote the subset of symbols in $\mathcal{S}$ ($\mathcal{F}$, $\mathcal{C}$) having arity $k \geq 0$ and let S^k (F^k, C^k) range over $\mathcal{S}^k$ ($\mathcal{F}^k$, $\mathcal{C}^k$).

The syntax of a core functional language of supercombinators is as follows.

$$\begin{aligned} T &::= x \mid S^k(T_1, \ldots, T_k) \mid \mathsf{let}\ x_1 = T_1, \ldots, x_n = T_n\ \mathsf{in}\ T \mid \\ &\qquad \mathsf{case}\ T\ \mathsf{of}\ P_1 \to E_1; \ldots; P_k \to E_k \\ P &::= C^k(x_1, \ldots, x_k) \end{aligned}$$

In $\mathsf{let}\ x_1 = T_1, \ldots, x_n = T_n\ \mathsf{in}\ T$, each x_i is bound in all T_i as well as in T (so this let is really a letrec). Function definitions are equations (one per function symbol F) of the form $F(\boldsymbol{x}) = T$.

Higher order functions are modelled in a standard way [13] by allowing partial applications of functions to less than their complete complement of arguments. This is done by requiring $\mathcal{F}$ to contain a binary application symbol @ and, for each $k \geq 1$, $F \in \mathcal{F}^k$, $0 \leq j < k$, requiring $\mathcal{C}^j$ to contain a constructor symbol C_F^j. Then $C_F^j(E_1, \ldots, E_j)$ represents the partial curried application $F E_1 \ldots E_j$ and the meaning of @ is given by equations of the form:

$$@(C_F^j(\boldsymbol{x}), y) = C_F^{j+1}(\boldsymbol{x}, y) \qquad\qquad @(C_F^{k-1}(\boldsymbol{x}), y) = F(\boldsymbol{x}, y).$$

To interpret programs in this language as graph rewriting systems, we use a restricted version of a form of equational term graph rewriting due to Ariola and Arvind [3]. This form of graph rewriting corrects a defect in the earlier work of Barendregt et al [5] concerning collapsing rules and cycles (see Ariola and Klop [2] for a discussion). The solution depends on the use of a special 'unboxed' constant $\circlearrowleft$, where the key feature of unboxed constants is that they can be freely substituted for variables in the terms used to represent graphs. We use the formalism of Ariola and Arvind [3], rather than that of Ariola and Klop [2], because we exploit the distinction between boxed and unboxed constants ($\top$ is unboxed) and because a variant of the graph model in the former allows us to define safety of our abstraction in a natural way.

Definition 2.1 (GRS Term). *Let* UC *denote some subset of* $\mathcal{C}^0 \cup \{\circlearrowleft\}$ *nominated as* the unboxed constants[1]. *We require that* $\circlearrowleft \in$ UC. *Let* v, w *range over* Var $\cup$ UC. *The GRS terms over signature* $\mathcal{S}$, *denoted* Ter($\mathcal{S}$), *are defined as follows:*

$$\begin{aligned} L, M, N &::= \mathsf{let}\ \{x_1 = E_1, \ldots, x_n = E_n\}\ \mathsf{in}\ v \\ E &::= M \mid v \mid S^k(v_1, \ldots, v_k) \mid \Omega \end{aligned}$$

[1] In [3] all nullary symbols are unboxed.

Here and in the remainder of the paper, writing $S^k(\boldsymbol{v})$ *should be taken to imply* $S^k \notin \mathrm{UC}$ *in the case that* $k = 0$.

In a term $M = \mathsf{let}\ \{\boldsymbol{x} = \boldsymbol{E}\}\ \mathsf{in}\ v$*, the order of bindings is irrelevant, all* x_i *are required to be distinct and are bound in all* E_i *as well as* v *(so* let *is a letrec). The sets* $\mathrm{FV}(M)$ *and* $\mathrm{BV}(M)$ *of the free and bound variables, respectively, of* M *are defined in the usual way.* $\mathrm{Ter}^\circ(\mathcal{S})$ *denotes the closed terms.*

The root of M*, denoted* $\mathrm{root}(M)$*, is* v.

In [3], a unique canonical form is defined for each term by flattening nested lets, removing 'trivial' bindings of the form $x = v$ and substituting v everywhere for x, and removing garbage. This is formalised in [3] as a strongly normalising reduction system on terms. For example, the reduction rule for flattening has the form:

$$\begin{aligned}&\mathsf{let}\ \{\boldsymbol{x} = \boldsymbol{E}, y = \mathsf{let}\ \{\boldsymbol{x}' = \boldsymbol{E}'\}\ \mathsf{in}\ w, \boldsymbol{x}'' = \boldsymbol{E}''\}\ \mathsf{in}\ v\\ &\to \mathsf{let}\ \{\boldsymbol{x} = \boldsymbol{E}, y = w, \boldsymbol{x}' = \boldsymbol{E}', \boldsymbol{x}'' = \boldsymbol{E}''\}\ \mathsf{in}\ v\end{aligned}$$

(assuming all bound variables distinct and the bound and free variables disjoint). Removal of trivial bindings is straightforward apart from those of the form $x = x$ (the substitution has no effect, so removal of the binding would result in x becoming free): these are dealt with by substituting $\circlearrowleft$ for x.

Definition 2.2 (Canonical Form). *A term* $\mathsf{let}\ \{\boldsymbol{x} = \boldsymbol{E}\}\ \mathsf{in}\ v$ *is* in canonical form *if:*

1. *each* E_i *is either* Ω *or of the form* $S^k(v_1, \dots, v_k)$ *;*
2. *each* x_i *is reachable from* v *(the term is garbage free).*

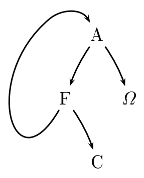
Let $\overline{M}$ *denote the result of flattening and removal of trivial bindings, and let* $\mathrm{GC}(N)$ *denote* N *with garbage removed. Then* $\mathrm{GC}(\overline{M})$ *is said to be* the canonical form for M*.*

Terms M *and* N *are said to be* graph equivalent*, written* $M \equiv N$*, if the canonical forms for* M *and* N *are the same up to renaming of bound variables.*

Canonical forms correspond in an obvious way to rooted labelled digraphs which are finite and connected. As an example, consider the term

$$\mathsf{let}\ \{x' = \mathsf{let}\ \{x = A(y, z),\ z = \Omega\}\ \mathsf{in}\ x,\ y = F(x', y'),\ y' = C\}\ \mathsf{in}\ x'.$$

Assuming $C \in \mathrm{UC}$, this has canonical form $\mathsf{let}\ \{x = A(y, z),\ y = F(x, C),\ z = \Omega\}\ \mathsf{in}\ x$, which directly corresponds to the following graph:



Predicates and relations defined on terms in canonical form are to be understood as being implicitly extended to arbitrary terms via their canonical forms. For example, the rooted homomorphism preorder $\leqslant$ (see Def. 2.4) is extended to arbitrary terms thus: $M \leqslant N$ iff $GC(\overline{M}) \leqslant GC(\overline{N})$.

The correspondence between canonical forms and graphs makes it possible to transpose many familiar graph definitions to terms in canonical form.

Definition 2.3 (Label, Vertex, Edge, Path). *Let* $M = \mathsf{let}\ \{\boldsymbol{x} = \boldsymbol{E}\}\ \mathsf{in}\ v$ *be in canonical form.*

1. *For* $x \in \mathrm{BV}(M)$ *such that* x *is not bound to* Ω*,* the label for x *is the unique* $S \in \mathcal{S}$ *such that* x *is bound to* $S(\boldsymbol{y})$*.*
2. The set of vertices of M*, denoted* $\mathrm{vert}(M)$*, is* $\mathrm{FV}(M) \cup \mathrm{BV}(M) \cup \mathrm{UC}$.
3. An edge[2] in M *is a pair* (x, v) *of vertices of* M *such that* $x = S^k(v_1, \dots, v_k)$ *is a binding in* M *with* $v = v_j$ *for some* $1 \leq j \leq k$*.*
4. *For* $n \geq 0$*,* a path of length n in M *is a sequence* $v_0 \cdots v_n$ *of* $n+1$ *vertices of* M*, such that, for every* $0 \leq i < n$*,* (v_i, v_{i+1}) *is an edge in* M*.*

Definition 2.4 (Homomorphism). *Given terms* M *and* N *in canonical form,* a homomorphism from M to N *is a map* $\sigma : \mathrm{vert}(M) \to \mathrm{vert}(N)$ *such that:*

1. $\sigma(v) = v$*, if* $v \notin \mathrm{BV}(M)$ *;*
2. *if* x *is bound to* $S^k(\boldsymbol{v})$ *in* M*, then* $\sigma(x)$ *is bound to* $S^k(\sigma(\boldsymbol{v}))$ *in* N*.*

(We write $\sigma(\boldsymbol{v})$ *to mean* $\sigma(v_1), \dots, \sigma(v_k)$*.) If, furthermore,* $\sigma(\mathrm{root}(M)) = \mathrm{root}(N)$*, then* σ *is called* a rooted homomorphism *and we write* $\sigma : M \leqslant N$*. We write* $M \leqslant N$ *to mean that there exists such a* σ*.*

Note that if x *is bound to* Ω *in* M*, it is not necessary to have* $\sigma(x)$ *bound to* Ω *in* N *(or, indeed, even to have* $\sigma(x) \in \mathrm{BV}(N)$*).*

It is easily verified that (rooted) homomorphisms compose and that the identity is a rooted homomorphism, hence $\leqslant$ is a preorder (which, in fact, induces $\equiv$).

We now introduce Ariola and Arvind's formalisation of the general notion of term graph rewrite rule.

Definition 2.5 (GRS Rewrite Rule). A GRS rewrite rule *(henceforth, just* rewrite rule*)* ρ *has the form:*

$$\frac{z = E_l,\ x_1 = E_1, \dots, x_n = E_n}{z \to E_r}$$

($n \geq 0$*) where, letting* $\mathrm{lhs}(\rho)$ *denote the term* $\mathsf{let}\ \{z = E_l,\ x_1 = E_1, \dots, x_n = E_n\}\ \mathsf{in}\ z$ *and letting* $\mathrm{rhs}(\rho)$ *denote* E_r*:*

1. $\mathrm{lhs}(\rho)$ *is in canonical form and does not contain* Ω *(thus* E_l *and each* E_i *must be of the form* $S^k(\boldsymbol{v})$*);*

[2] The notions of edge and path defined here contain less information than the standard ones but are sufficient for our purposes. The standard ones can be defined in similar fashion.

2. $\mathrm{FV}(\mathrm{rhs}(\rho)) \subseteq \mathrm{FV}(\mathrm{lhs}(\rho)) \cup \mathrm{BV}(\mathrm{lhs}(\rho))$.

The closure of $\mathrm{lhs}(\rho)$ *is* $\mathsf{let}\ \{\boldsymbol{y} = \boldsymbol{\Omega}, \boldsymbol{x} = \boldsymbol{E}, z = E_l\}\ \mathsf{in}\ z$ *where* $\boldsymbol{y} = \mathrm{FV}(\mathrm{lhs}(\rho))$.

Note that the function/constructor distinction plays no role in the above definition. This notion of rewrite rule is actually far too general for our purposes and so below (Def. 2.8) we define a restricted form which is adequate for the core functional language described above.

Definition 2.6 (Redex). *Let P be a set of rewrite rules. Given term M in canonical form,* a P-redex in M *is a pair (ρ, σ), where ρ is in P and σ is a homomorphism from the closure of* $\mathrm{lhs}(\rho)$ *to M.*

Let L be the closure of $\mathrm{lhs}(\rho)$. *Then* $\sigma(\mathrm{root}(L))$ *(which is necessarily in* $\mathrm{BV}(M)$*) is known as* the root of the redex. *Note that the domain of σ contains* $\mathrm{FV}(\mathrm{rhs}(\rho))$.

Definition 2.7 (Rewrite). *Let P be a set of rewrite rules, let $M = \mathsf{let}\ \{x_1 = E_1, \ldots, x_n = E_n\}\ \mathsf{in}\ v$ and let $\Delta = (\rho, \sigma)$ be a P-redex in M. Let x_i be the root of Δ. M is said to* rewrite to $\mathrm{GC}(\overline{N})$ by Δ *if $N = M[x_i \leftarrow (\mathrm{rhs}(\rho))^\sigma]$, where E^σ denotes the application of σ as a substitution on the free variables of E, and $M[x_i \leftarrow E]$ denotes the term* $\mathsf{let}\ \{x_1 = E_1, \ldots, x_i = E, \ldots, x_n = E_n\}\ \mathsf{in}\ v$.

If M rewrites to M' by some P-redex Δ, we write $M \xrightarrow{P} M'$. We write $\xrightarrow{P}\!\!\!\!\rightarrow$ for the transitive reflexive closure of $\xrightarrow{P}$ and we write $\xrightarrow{P}_{=}$ for the reflexive closure.

Definition 2.8 (Simple Functional GRS). *A rewrite rule ρ is said to be* left-linear *if the graph represented by* $\mathrm{lhs}(\rho)$ *is a tree.* A simple functional GRS *over $\mathcal{S}$ is a set* P *of left-linear rewrite rules of a limited form, namely, for each function symbol $F \in \mathcal{F}$:*

1. either *a single rewrite rule of the form*

$$\frac{z = F(\boldsymbol{y})}{z \to E}$$

(note that $\boldsymbol{y}$ is a vector of variables, not arbitrary vertices);

2. or *a family of n rewrite rules, each of either of the following two forms:*

$$\frac{z = F(y_0, \boldsymbol{y}),\ y_0 = C_i(\boldsymbol{x}_i)}{z \to E_i} \qquad \frac{z = F(C_i, \boldsymbol{y})}{z \to E_i},$$

with each C_i a distinct constructor symbol, $1 \leq i \leq n$.

When function symbol F is defined by a family of rules as in (2) we say that F is pattern-matching.

Any program written in the core functional language described at the beginning of this section can be translated in a straightforward manner into a simple functional GRS. In essence, each use of `case` is translated to a pattern-matching function definition (in general, this will involve lifting the locally bound variables of a `case` to the top level, as in lambda-lifting). See Barendsen and Smetsers [6] for examples.

Say that $\xrightarrow{\alpha}$ *subcommutes* with $\xrightarrow{\beta}$ if $\xleftarrow{\alpha};\xrightarrow{\beta} \subseteq \xrightarrow{\beta}_{=};{}_{=}\xleftarrow{\alpha}$. A rewrite relation is said to be *subcommutative* if it subcommutes with itself. It is a standard result that subcommutativity implies confluence (see eg [14]).

Proposition 2.9. *The rewrite relation generated by a simple functional GRS is subcommutative, hence confluent, up to $\equiv$.*

Proof. The restrictions placed on the form of rules allowed in a simple functional GRS ensure that the rules are non-overlapping (non-interfering in the terminology of [3]). Subcommutativity is immediate by Theorem 4.8 in [3]. $\square$

3 Abstracting a Simple Functional GRS

In this section we define abstract GRS terms as a superset of the ordinary GRS terms and go on to define an abstraction relation on abstract terms. We then define a way of extending a simple functional GRS on ordinary terms to one on abstract terms.

Definition 3.1 (Abstract GRS Term). An abstract GRS term over $\mathcal{S}$ *is a GRS term over the signature $\mathcal{S}^\# = \mathcal{F} \cup \mathcal{C}^\#$, where $\mathcal{C}^\#$ is $\mathcal{C}$ plus the following (assumed new) constructor symbols:*

1. *$\top$, of arity 0, $\top \in \mathrm{UC}$;*
2. *for each $k \geq 1$, the union symbol $\oplus^k$ of arity k (we will often omit the superscript k).*

Let H, J, K range over abstract terms.

Note that the abstract terms over $\mathcal{S}$ include the terms over $\mathcal{S}$ as a special case: $\mathrm{Ter}(\mathcal{S}) \subset \mathrm{Ter}(\mathcal{S}^\#)$.

Our formalisation of the meaning of $\oplus$ is provided below by the notions of choice set and weak homomorphism. The first of these goes to heart of our graph-based approach, showing how each vertex labelled $\oplus$ is used to represent a set of non-$\oplus$ vertices.

Definition 3.2 (Choice Set). *Let H be an abstract term in canonical form. Given $v \in \mathrm{vert}(H)$,* a choice path for v *is a path $v = v_0, \dots, v_n$ in M such that, for all $0 \le i < n$, v_i is labelled $\oplus$. Let* the set of definite vertices of H*, denoted* $\mathrm{dvert}(H)$*, be the set $\{v \in \mathrm{vert}(H) \,|\, v$ is not labelled $\oplus\}$.*

The choice set *for $v \in \mathrm{vert}(H)$, denoted $\chi(v)$, is the smallest set $X \subseteq$* $\mathrm{dvert}(H)$ *such that:*

1. *$\circlearrowright \in X$ if there is a choice path $v = v_0 \cdots v_i \cdots v_n = v_i$ with $0 \le i < n$;*
2. *$v' \in X$ if there is a choice path $v = v_0 \cdots v_n = v'$ and v' is not labelled $\oplus$.*

Lemma 3.3. *If $v \in \mathrm{dvert}(H)$ then $\chi(v) = \{v\}$. If x is bound to $\oplus^k(\boldsymbol{v})$ in H, then $\chi(x) = \chi(v_1) \cup \cdots \cup \chi(v_k)$.*

The definition of homomorphism (Def. 2.4) requires that all labels are preserved. Given the role played by $\oplus$ labels, it is inappropriate to require them to be strictly preserved when comparing abstract graphs. The following provides a weakening of the notion of homomorphism which reflects the special status of $\oplus$:

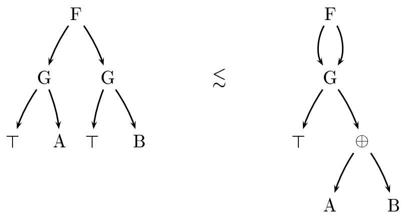
Definition 3.4 (Weak Homomorphism). *Given abstract terms H and J in canonical form,* a weak homomorphism from H to J *is a map $\sigma : \mathrm{dvert}(H) \to \mathrm{dvert}(J)$ such that:*

1. *$\sigma(v) = v$, if $v \notin \mathrm{BV}(H)$;*
2. *if x is bound to $S^k(\boldsymbol{v})$ in H, $S^k \neq \oplus^k$, then $\sigma(x)$ is bound to $S^k(\boldsymbol{w})$ in J and $\forall i, 1 \leq i \leq k. \forall v \in \chi(v_i). \sigma(v) \in \chi(w_i)$.*

If, furthermore, $\forall v \in \chi(\mathrm{root}(H)). \sigma(v) \in \chi(\mathrm{root}(J))$, then σ is called a rooted weak homomorphism *and we write $\sigma : H \lesssim J$. We write $H \lesssim J$ to mean that there exists such a σ.*

It is easily verified that (rooted) weak homomorphisms compose and that the identity is a rooted weak homomorphism, hence $\lesssim$ is a preorder.

Example 3.5 (Weak Homomorphism).



There is a unique rooted weak homomorphism σ which maps the nodes labelled F, G, A, B on the left to their counterparts on the right (we assume neither A or B is unboxed). Since $\top$ is unboxed, the definition of weak homomorphism forces $\sigma(\top) = \top$. Note that, like a homomorphism, a weak homomorphism cannot decrease sharing. Hence there is no weak homomorphism from right to left.

The notions of choice set and weak homomorphism capture the meaning of $\oplus$ but they do not say anything about the meaning of $\top$. To do this we define a relation between abstract terms as follows:

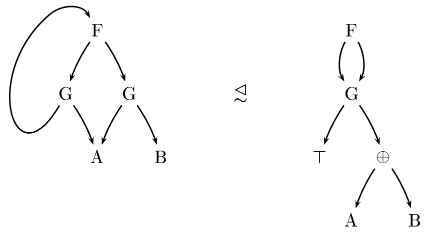
Definition 3.6 (Abstraction Morphism). *Given abstract terms H and J, both in canonical form, let $\tau : \mathrm{dvert}(H) \to \mathrm{dvert}(J)$. The relation $\leq_\tau \subseteq \mathrm{dvert}(H) \times \mathrm{dvert}(J)$ is defined as follows: $v \leq_\tau w$ iff $w = \tau(v) \vee w = \top$. Then τ is* an abstraction morphism from H to J *if:*

1. *$\tau(v) = v$, if $v \notin \mathrm{BV}(H)$;*
2. *if x is bound to $S^k(\boldsymbol{v})$ in H, $S^k \neq \oplus^k$, then* either *$\tau(x) = \top$,* or *$\tau(x)$ is bound to $S^k(\boldsymbol{w})$ in J and $\forall i, 1 \leq i \leq k. \forall v \in \chi(v_i). \exists w \in \chi(w_i). v \leq_\tau w$.*

If, furthermore, $\forall v \in \chi(\mathrm{root}(H)).\, \exists w \in \chi(\mathrm{root}(J)).\, v \leq_\tau w$*, then* τ *is called* a rooted abstraction morphism *and we write* $\tau : H \underset{\sim}{\lhd} J$*. We say that* H is abstracted by J*, written* $H \underset{\sim}{\lhd} J$*, if there exists such a* τ*.*

It is easily verified that (rooted) abstraction morphisms compose and that the identity is a rooted abstraction morphisms, hence $\underset{\sim}{\lhd}$ is a preorder. Clearly, all terms are abstracted by $\top$.

Example 3.7 (Abstraction).



There is a unique rooted abstraction morphism τ from left to right. Note that τ must map the two vertices labelled G on the left to a single vertex on the right.

Note the similarity between the definition of abstraction morphism and the definition of weak homomorphism. In fact, abstraction can be seen as a conservative extension of weak homomorphism, which in turn can be seen as a conservative extension of $\leqslant$, as shown by the following lemma:

Lemma 3.8. *1.* $\leqslant \subseteq \lesssim \subseteq \underset{\sim}{\lhd}$.
2. $\underset{\sim}{\lhd}$ *restricts to* $\lesssim$ *on terms not containing* $\top$.
3. $\lesssim$ *restricts to* $\leqslant$ *on terms not containing* $\oplus$.

We now define a simple functional GRS P# for rewriting abstract graphs. The main burden of our remaining technical development is to define an appropriate notion of safety and to show that rewriting abstract graphs using P# safely approximates the rewriting of ordinary graphs using P. Unlike previous approaches (eg [17, 22]), we do not define an abstract evaluator based on an abstract notion of pattern matching. Instead, we provide rewrite rules which apply whenever $\oplus$ or $\top$ occur in the pattern position for a pattern-matching function.

Definition 3.9 (Extension of a Simple Functional GRS). *Let* P *be a simple functional GRS. Then* the extension of P to abstract terms*, denoted* P#*, is* P *extended with additional rules for each pattern-matching* F *as follows: let the rule set for* F *in* P *comprise* n *rules, each of either of the two forms:*

$$\frac{z = F(y_0, \boldsymbol{y}),\, y_0 = C_i(\boldsymbol{x}_i)}{z \to E_i} \qquad\qquad \frac{z = F(C_i, \boldsymbol{y})}{z \to E_i},$$

We add to P# *the following rules:*

1.

$$\frac{z = F(y_0, \boldsymbol{y}),\ y_0 = \oplus^k(z_1, \ldots, z_k)}{z \to H}$$

for each $\oplus^k$, *where* H *is the term*

$$\mathsf{let}\ \{z' = \oplus^k(z'_1, \ldots, z'_k),\ z'_1 = F(z_1, \boldsymbol{y}),\ \ldots,\ z'_k = F(z_k, \boldsymbol{y})\}\ \mathsf{in}\ z'$$

(z' *and each* z_i, z'_i *fresh). We call these* the $\oplus$-lifting rules*.*

2.

$$\frac{z = F(\top, \boldsymbol{y})}{z \to J}$$

where J *is the term*[3]

$$\mathsf{let}\ \{z' = \oplus^n(z_1, \ldots, z_n),\ z_1 = E_1^\sigma,\ \ldots,\ z_n = E_n^\sigma\}\ \mathsf{in}\ z',$$

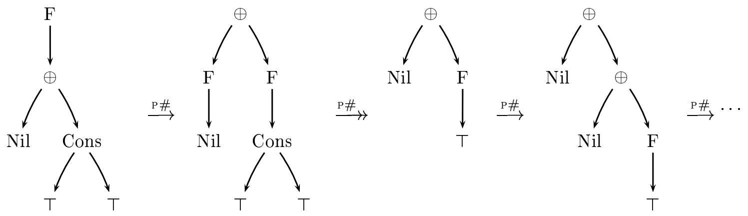
(each z_i *fresh) and* $\sigma(x) = \begin{cases} x & \text{if } x \in \boldsymbol{y} \\ \top & \text{otherwise.} \end{cases}$ *We call these the* $\top$ *rules.*

Note that P# is itself a simple functional GRS. The rewrite relation $\xrightarrow{\text{P\#}}\!\!\!\!\rightarrow$ on $\text{Ter}(\mathcal{S}^\#)$ is easily seen to be a conservative extension of $\xrightarrow{\text{P}}\!\!\!\!\rightarrow$ on $\text{Ter}(\mathcal{S})$.

Example 3.10 (Rewriting by P#*).* Suppose F is defined in P by the pair of rules:

$$\frac{z = \text{F}(\text{Nil})}{z \to \text{Nil}} \qquad \frac{z = \text{F}(y),\ y = \text{Cons}(x_1, x_2)}{z \to \text{F}(x_2)}$$

Then we have the P# rewrite sequence:



It is clear from this example that not all abstract graphs have a normal form with respect to $\xrightarrow{\text{P\#}}$.

[3] J will be different for different choices of ordering $i = 1 \ldots n$ on the rules for F. However, since $\oplus^n$ is a commutative operator with respect to the semantic precongruence induced by $\precsim$ (see Sect. 5), the differences are not semantically significant.

4 The Graph Model

We require a safety result which states that rewriting an abstract term H by the rules in $\textsc{p}\#$ gives information which is valid for all terms M such that $M \precsim\!\!\lhd H$. We state our safety property in terms of a variant of the direct approximation style semantics defined in [3]. We begin by defining what we regard as the directly observable part of a closed term.

Definition 4.1 (Directly Observable Approximation). *Let $\mathcal{S} = \mathcal{F} \cup \mathcal{C}$ be a signature. Given $M \in \mathrm{Ter}^{\circ}(\mathcal{S})$, let $U_M = \{z \in \mathrm{BV}(M) \mid z$ does not have a label in $\mathcal{C}\}$. The map $\omega : \mathrm{Ter}^{\circ}(\mathcal{S}) \to \mathrm{Ter}^{\circ}(\mathcal{C})$ is defined by $\omega(M) = M[U_M \leftarrow \Omega]$. $\omega(M)$ is known as* the directly observable approximation of M*.*

The following lemma collects together some basic facts about ω:

Lemma 4.2. *Let H, J be closed abstract terms over $\mathcal{S}$. Let $\longrightarrow$ be the rewrite relation generated by any simple functional GRS over $\mathcal{S}^{\#}$.*

1. $\omega(H) \leqslant H$.
2. $\omega(H) \leqslant J \Rightarrow \omega(H) \leqslant \omega(J)$.
3. $\omega(H) \lesssim J \Rightarrow \omega(H) \lesssim \omega(J)$.
4. $\omega(H) \precsim\!\!\lhd J \Rightarrow \omega(H) \precsim\!\!\lhd \omega(J)$.
5. $H \longrightarrow\!\!\!\!\!\rightarrow J \Rightarrow \omega(H) \leqslant \omega(J)$.

Definition 4.3 (Context). A context over signature $\mathcal{S}$ *is a GRS term* let $\{x_1 = E_1, \ldots, x_n = E_n\}$ in v *except that (at most) one E_i is allowed to be $\Box$. Let $\mathrm{Ctx}(\mathcal{S})$ denote the contexts over signature $\mathcal{S}$.*

Let $\mathfrak{C}, \mathfrak{D}$ range over contexts. If $\mathfrak{C}$ is a context containing the binding $x_i = \Box$, the result of placing a term M in $\mathfrak{C}$*, written $\mathfrak{C}[M]$, is the term $\mathfrak{C}[x_i \leftarrow M]$. If $\Box$ does not occur in $\mathfrak{C}$ then $\mathfrak{C}[M] = \mathfrak{C}$.*

Definition 4.4 (Compatible Relation). *We say that a binary relation R on terms is* compatible *if, for all contexts $\mathfrak{C}$, for all terms M, N, if $M \; R \; N$ then $\mathfrak{C}[M] \; R \; \mathfrak{C}[N]$.*

A compatible preorder is said to be a precongruence*. A compatible equivalence relation is said to be a* congruence*.*

It is standard that if R is a preorder (precongruence) then $R \cap R^{-1}$ is an equivalence relation (congruence).

Lemma 4.5. *The following are compatible:*

1. *The transitive reflexive closure of any rewrite relation generated by a set of GRS rewrite rules;*
2. $\leqslant$, $\lesssim$ *and* $\precsim\!\!\lhd$.

Definition 4.6 (Observational Preorder and Observational Equivalence). *Let $\textsc{p}$ be a simple functional GRS over signature $\mathcal{S}$ and let $M, N \in \mathrm{Ter}(\mathcal{S})$ be in canonical form. We say that M* is observably less than N with respect to $\textsc{p}$, *written* $M \sqsubsetsim_{\mathrm{P}} N$*, if, for all contexts* $\mathfrak{C} \in \mathrm{Ctx}(\mathcal{S})$ *such that* $\mathfrak{C}[M]$ *and* $\mathfrak{C}[N]$ *are closed,* $\forall M' \overset{\mathrm{P}}{\twoheadleftarrow} \mathfrak{C}[M].\, \exists N' \overset{\mathrm{P}}{\twoheadleftarrow} \mathfrak{C}[N].\, \omega(M') \leqslant \omega(N')$*. We say that* M and N are observationally equivalent with respect to P, *written* $M \cong_{\mathrm{P}} N$*, if* $M \sqsubsetsim_{\mathrm{P}} N$ *and* $N \sqsubsetsim_{\mathrm{P}} M$*.*

We will omit the P *subscripts when it is not confusing to do so.*

It is easily verified that $\sqsubsetsim$ is a preorder (hence $\cong$ is an equivalence relation). Furthermore, because the definition of $\sqsubsetsim$ quantifies over all contexts, it follows directly that $\sqsubsetsim$ is in fact a precongruence (hence $\cong$ is a congruence). The following result is an easy consequence of Lemma 4.2(5) plus confluence:

Proposition 4.7. *If* $M \overset{\mathrm{P}}{\twoheadrightarrow} N$ *then* $M \cong_{\mathrm{P}} N$*.*

5 Safety

The previous section defines a semantic precongruence, $\sqsubsetsim$, based on the rooted graph homomorphism preorder $\leqslant$ and rewriting by a simple functional GRS P. In this section we define a natural analogue of $\sqsubsetsim$ for abstract terms, based on the abstraction preorder $\trianglelefteqsim$ and rewriting by P#. As we show below, this 'abstract' precongruence is sound with respect to $\sqsubsetsim$, in the sense that it restricts to a subrelation of $\sqsubsetsim$ on ordinary terms.

Definition 5.1 (Semantic Extension). *Let* $\mathcal{S} = \mathcal{F} \cup \mathcal{C}$*, let* R *be a binary relation on* $\mathrm{Ter}^{\circ}(\mathcal{C})$ *and let* P *be a simple functional GRS over* $\mathcal{S}$*.* The semantic extension of R with respect to P, *denoted* $[R]_{\mathrm{P}} \subseteq \mathrm{Ter}(\mathcal{S}) \times \mathrm{Ter}(\mathcal{S})$ *is defined thus:* $M\ [R]_{\mathrm{P}}\ N$ *iff for all contexts* $\mathfrak{C} \in \mathrm{Ctx}(\mathcal{S})$ *such that* $\mathfrak{C}[M]$ *and* $\mathfrak{C}[N]$ *are closed,* $\forall M' \overset{\mathrm{P}}{\twoheadleftarrow} \mathfrak{C}[M].\, \exists N' \overset{\mathrm{P}}{\twoheadleftarrow} \mathfrak{C}[N].\, \omega(M')\ R\ \omega(N')$*.*

It is easily seen that, if R is a preorder then so is $[R]_{\mathrm{P}}$, for any P. Note that $\sqsubsetsim_{\mathrm{P}}$ is $[\leqslant]_{\mathrm{P}}$ (where $\leqslant$ is implicitly restricted to $\mathrm{Ter}^{\circ}(\mathcal{C}) \times \mathrm{Ter}^{\circ}(\mathcal{C})$).

Proposition 5.2. *Let* P *be a simple functional GRS over* $\mathcal{S}$*. Then*

$$[\trianglelefteqsim]_{\mathrm{P}\#} \restriction \mathrm{Ter}(\mathcal{S}) \times \mathrm{Ter}(\mathcal{S}) \subseteq \sqsubsetsim_{\mathrm{P}}$$

(where $R \restriction X$ *denotes the restriction of* R *to* X*).*

Proof. See Appendix.

Remark 5.3. We conjecture that $[\trianglelefteqsim]_{\mathrm{P}\#}$ is in fact a conservative *extension* of $[\leqslant]_{\mathrm{P}}$ (ie, that $\subseteq$ can be strengthened to $=$ in the statement of the proposition). In the current setting it is not clear that this is a useful property.

This result shows that, in principle, P# can be used to predict the semantics of terms with respect to P. This is not obviously useful, since $\overset{\mathrm{P}\#}{\twoheadrightarrow}$ is no easier to compute than $\overset{\mathrm{P}}{\twoheadrightarrow}$. In particular, it is easy to see that, in general, $[\trianglelefteqsim]_{\mathrm{P}\#}$ will be undecidable. Clearly we must settle for a sound method - one which allows us to decide a subrelation of $[\lesssim\!\!\lhd]_{P\#}$, which, by the proposition, in turn decides a subrelation of $\sqsubseteq\!\!\!\sim_{P}$. The key to developing such a method is to show that $\lesssim\!\!\lhd$ is itself a safe approximation to $[\lesssim\!\!\lhd]_{P\#}$. Proving this directly is not straightforward and we postpone it to Sect. 6 (Corollary 6.6).

Definition 5.4 (Safety). *Let* P *be a simple functional GRS over* $\mathcal{S}$ *and let* R *be a binary relation on* $\mathrm{Ter}(\mathcal{S}^{\#})$. *We say that* R is safe *if* $R \subseteq [\lesssim\!\!\lhd]_{P\#}$.

Note that, because $[\lesssim\!\!\lhd]_{P\#}$ is a preorder, R is safe iff R^* is safe. The following results shows this notion of safety to be adequate for strictness analysis:

Proposition 5.5. *Let* P *be a simple functional GRS over* $\mathcal{S}$. *Let* $\xrightarrow{\mathrm{AR}}$ *be some notion of abstract reduction for* P *(that is,* $\xrightarrow{\mathrm{AR}}$ *is a binary relation on* $\mathrm{Ter}(\mathcal{S}^{\#})$*). If* $\lesssim\!\!\lhd$ *is safe and* $\xrightarrow{\mathrm{AR}}$ *is safe then* $H \xrightarrow{\mathrm{AR}}\!\!\!\!\!\rightarrow \Omega \Rightarrow M \cong_{P} \Omega$ *for all* $M \lesssim\!\!\lhd H$.

Note that M need not be closed. This result entails, for example, that if $\lesssim\!\!\lhd$ and $\xrightarrow{\mathrm{AR}}$ are safe, then $F(\Omega, \top) \xrightarrow{\mathrm{AR}}\!\!\!\!\!\rightarrow \Omega$ implies $F(\Omega, x) \cong_{P} \Omega$, with x free. (This is clearly an analogue of the usual denotational definition of strictness in the first argument of a binary function: $\forall d.\, f(\bot, d) = \bot$. Formalising this would involve the elaboration of a denotational model for P but it is easy to see that trivially non-terminating programs, such as $\mathsf{let}\ \{z = L(0)\}\ \mathsf{in}\ z$ where $L(x) \to L(x)$, are $\cong_{P} \Omega$.)

6 Abstract Reduction

We assume given some simple functional GRS P over a signature $\mathcal{S} = \mathcal{F} \cup \mathcal{C}$.

We define a relation $\xrightarrow{\mathrm{AR}}$, and show that it is safe. $\xrightarrow{\mathrm{AR}}$ is defined as the union of a number of subrelations, R_i, each of which has the following property:

$$H\ R_i\ J \Rightarrow \omega(H) \lesssim\!\!\lhd \omega(J) \tag{1}$$

This allows us to establish safety of $\xrightarrow{\mathrm{AR}}$ via a commutativity property:

Lemma 6.1. *Let* $\xrightarrow{\mathrm{AR}} = \bigcup_{i \in I} R_i$ *with each* $R_i \subseteq \mathrm{Ter}(\mathcal{S}^{\#}) \times \mathrm{Ter}(\mathcal{S}^{\#})$. *If each* R_i *satisfies (1) and each* R_i^* *is compatible, then* $\xrightarrow{\mathrm{AR}}$ *is safe if it commutes with* $\xrightarrow{P\#}$.

Proof. See Appendix.

It follows easily from Prop. 4.7 that $\xrightarrow{P\#}$ is safe, so it is reasonable to include $\xrightarrow{P\#}$ in $\xrightarrow{\mathrm{AR}}$. Now consider Example 3.10. The function F is clearly strict but we do not have $\mathrm{F}(\Omega) \xrightarrow{P\#}\!\!\!\!\!\rightarrow \Omega$. We would like to add an 'Ω-lifting rewrite rule' $F(\Omega, \boldsymbol{y}) \to \Omega$ for every pattern-matching F. Unfortunately this is not well-defined, since Ω is disallowed in the left hand sides of rewrite rules, but the idea motivates the following definition:

Definition 6.2. *Let H, J be abstract terms in canonical form. Then $H \xrightarrow{\Omega_{\mathrm{L}}} \mathrm{GC}(\overline{J})$ iff there exist pattern-matching F^k, bound variables $z, y_0 \in \mathrm{BV}(H)$, and vertices $v_1, \ldots, v_{k-1} \in \mathrm{vert}(H)$, such that y_0 is bound to Ω in H and z is bound to $F^k(y_0, v_1, \ldots, v_{k-1})$ in H and $J = H[z \leftarrow \Omega]$.*

Lemma 6.3. *1. $\overset{\Omega_{\mathrm{L}}}{\twoheadrightarrow}$ is compatible and $\xrightarrow{\Omega_{\mathrm{L}}}$ satisfies (1).*
2. $\overset{\Omega_{\mathrm{L}}}{\twoheadrightarrow}$ subcommutes with itself and with $\overset{\mathrm{P}}{\twoheadrightarrow}$ for any simple functional GRS P.

Nöcker's analysis method amounts to defining $\xrightarrow{\mathrm{AR}}$ as $\xrightarrow{\mathrm{P}\#} \cup \xrightarrow{\Omega_{\mathrm{L}}}$ plus some 'heuristic' reductions which are used to force termination. There are two forms of heuristic described in [17]. The first of these dynamically coarsens the abstract graph by replacing some parts of it by $\top$ (this may be done when some bound on the number of rewrite steps is reached). The second form performs checks to detect looping behaviour in the abstract evaluator and responds by either replacing parts of the graph by Ω or introducing a cycle into the graph. In addition, Nöcker's analyser applies a number of reduction rules to simplify subgraphs involving $\oplus$ (for example: $\oplus(x, y, y) \rightarrow \oplus(x, y)$).

We do not account for the heuristics based on loop-detection in the current paper. Replacement by $\top$ and the simplification of $\oplus$ subgraphs are subsumed by including $\underset{\sim}{\trianglelefteq}$ in $\xrightarrow{\mathrm{AR}}$. Thus we arrive at the following notion of abstract reduction:

Definition 6.4. $\xrightarrow{\mathrm{AR}} = \xrightarrow{\mathrm{P}\#} \cup \xrightarrow{\Omega_{\mathrm{L}}} \cup \underset{\sim}{\trianglelefteq}$.

Theorem 6.5. *$\xrightarrow{\mathrm{AR}}$ commutes with $\xrightarrow{\mathrm{P}\#}$.*

Proof. See Appendix.

Corollary 6.6. *$\underset{\sim}{\trianglelefteq}$ and $\xrightarrow{\mathrm{AR}}$ are safe.*

Proof. The subrelations $\underset{\sim}{\trianglelefteq}$, $\xrightarrow{\mathrm{P}\#}$, $\xrightarrow{\Omega_{\mathrm{L}}}$ are each compatible (Lemma 4.5, Lemma 6.3(1)) and satisfy (1) (Lemma 4.2, Lemma 6.3(1)) thus safety of $\xrightarrow{\mathrm{AR}}$ follows by Lemma 6.1. Safety of $\underset{\sim}{\trianglelefteq}$ is implied by safety of $\xrightarrow{\mathrm{AR}}$, since $\underset{\sim}{\trianglelefteq} \subseteq \xrightarrow{\mathrm{AR}}$. □

7 Conclusions and Further Work

Nöcker's method of abstract reduction is an intuitively appealing approach to program analysis which is quite unlike any of the other approaches to strictness analysis which have been proposed in the literature. In this paper we have succeeded for the first time in formalising abstract reduction and proving its correctness without any restrictions on the class of abstract graphs. Future work will address the correctness of the heuristics based on loop-detection, which are used in the Clean strictness analyser to force termination. We do not believe that this will require a change to our formalisation. The main technical requirement is to establish a standardisation result, allowing semantic properties to be inferred from cyclic behaviour along particular (rather than all) reduction paths.

Being tailored to strictness analysis, the approach that we have taken is rather *ad hoc*. Ideally, we would like to establish a general framework and explore the application of abstract reduction to other program analyses. To start with, it may be relatively easy to adapt the results of this paper to binding time analysis. Beyond that, a striking aspect of the notion of abstraction developed in this paper is that it is sensitive to differences in sharing. Information about sharing properties can be very useful to language implementors [21, 6, 24] and it will be interesting to see if a form of sharing analysis can be developed by adapting our technique.

Another future direction would be to develop similar techniques for other forms of operational semantics. It would be an advantage to have a framework which applied directly to lambda calculi, rather than having to work with supercombinators. One possibility would be to use the rewriting theory of cyclic lambda calculi developed by Ariola and Blom [4]. An alternative would be to use an abstract machine approach, such as that used by Moran and Sands [19].

Finally, it remains open whether abstract reduction can be formalised within the framework of abstract interpretation. We can draw the following analogies: Collecting interpretation $\sim$ graphs built using $\oplus$, ordered by $\lesssim$, plus the $\oplus$-lifting rules; Abstraction of the collecting semantics $\sim$ graphs built using $\top$, ordered by $\lhd$, plus the $\top$ rules; Widening $\sim$ heuristics modelled by $\underset{\sim}{\lhd}$. But many unanswered questions remain.

We would like to thank Richard Kennaway, Matthias Felleisen, Thomas Jensen and the anonymous referees for their suggestions and comments.

References

[1] S. Abramsky and C.L. Hankin, editors. *Abstract Interpretation of Declarative Languages*. Computers and Their Applications. Ellis Horwood, 1987.

[2] Z. Ariola and J. W. Klop. Equational term graph rewriting. *Fundamentae Informatica*, 26(3,4):207–240, June 1996.

[3] Zena Ariola and Arvind. Properties of a first-order functional language with sharing. *Theoretical Computer Science*, 146(1–2):69–108, July 1995.

[4] Zena Ariola and Stefan Blom. Cyclic lambda calculi. In *Proc. TACS'97*. Springer-Verlag, February 1997. LNCS 1281.

[5] H.P. Barendregt, M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer, and M.R. Sleep. Term graph rewriting. In *Proc. PARLE 87*, volume 2, pages 141–158. Springer-Verlag LNCS 259, Eindhoven, The Netherlands, June 1987.

[6] Erik Barendsen and Sjaak Smetsers. Uniqueness typing for functional languages with graph rewriting semantics. *Mathematical Structures in Computer Science*, 6:579–612, 1996.

[7] Tyng-Ruey Chuang and Benjamin Goldberg. A syntactic approach to fixed point computation on finite domains. In *Proc. 1992 ACM Conference on LISP and Functional Programming*. ACM, June 1992.

[8] D. J. Clark and C. Hankin. A lattice of abstract graphs. In M. Bruynooghe and J Penjam, editors, *Proc. Programming Language Implementation and Logic Programming*, pages 318–331, Tallinn, Estonia, August 1993. Springer-Verlag LNCS 714.

[9] David J. Clark. *Term Graph Rewriting and Event Structures.* PhD thesis, Imperial College, University of London, 1996.

[10] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixed points. In *Proc. Fourth ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, 1977.

[11] Andrew D. Gordon and Andrew M. Pitts, editors. *Higher Order Operational Techniques in Semantics.* Publications of the Newton Institute. Cambridge University Press, 1998.

[12] E. Goubault and C. L. Hankin. A lattice for the abstract interpretation of term graph rewriting systems. In *Term Graph Rewriting, theory and practice*, pages 131–140. John Wiley and Sons Ltd., 1993.

[13] J. R. Kennaway, J. W. Klop, M. R. Sleep, and F. J. de Vries. Comparing curried and uncurried rewriting. *J. Symbolic Computation*, 21:15–39, 1996.

[14] Jan Willem Klop. Term rewriting systems. In Samson Abramsky, Dov M. Gabbay, and Tom Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, New York, 1992.

[15] F. Nielson, H. R. Nielson, and C. L. Hankin. *Principles of Program Analysis.* Springer, 1999.

[16] E. Nöcker. Strictness analysis based on abstract reduction of term graph rewrite systems. In *Proc. Workshop on Implementation of Lazy Functional Languages*, 1988.

[17] E. Nöcker. Strictness analysis using abstract reduction. In *Proc. Conference on Functional Programming Languages and Computer Architectures (FPCA '93)*, Copenhagen, 1993. ACM Press.

[18] E. Nöcker. *Efficient Functional Programming.* PhD thesis, Department of Computer Science, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, 1994.

[19] David Sands and Andrew Moran. Improvement in a lazy context: an operational theory for call-by-need. In *Proc. Twenty-sixth ACM Symposium on Principles of Programming Languages*, San Antonio, Texas, January 1999.

[20] M. Schütz, M. Schmidt-Schauß and S. E. Panitz. Strictness analysis by abstract reduction using a tableau calculus. In A. Mycroft, editor, *Proc. Static Analysis, Second International Symposium, SAS '95*, number 983 in Lecture Notes in Computer Science, pages 348–365, Glasgow, UK, September 1995. Springer Verlag.

[21] David N. Turner, Philip Wadler, and Christian Mossin. Once upon a type. In *Proc. 7'th International Conference on Functional Programming and Computer Architecture*, San Diego, California, June 1995.

[22] M. C. J. van Eekelen, E. Goubault, C. L. Hankin, and E. Nöcker. Abstract reduction: Towards a theory via abstract interpretation. In *Term Graph Rewriting, theory and practice*, pages 117–130. John Wiley and Sons Ltd., 1993.

[23] Vincent van Oostrom. *Confluence for Abstract and Higher-Order Rewriting.* PhD thesis, Vrije Universiteit, Amsterdam, 1994.

[24] Keith Wansbrough and Simon Peyton Jones. Once upon a polymorphic type. In *Proc. Twenty-sixth ACM Symposium on Principles of Programming Languages*, San Antonio, Texas, January 1999.

[25] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38–94, 15 November 1994.

A Appendix: proof outlines for key results

Proof: Proposition 5.2

Suppose $M\ [\precsim\!\!\!\lhd]_{P\#}\ N$ with $M, N \in \mathrm{Ter}(\mathcal{S})$ and suppose $M' \xleftarrow{P}\!\!\!\!\leftarrow \mathfrak{C}[M]$, with $\mathfrak{C} \in \mathrm{Ctx}(\mathcal{S})$ a closing context for M and N. We must show that there exists $N' \xleftarrow{P}\!\!\!\!\leftarrow \mathfrak{C}[N]$ such that $\omega(M') \leqslant \omega(N')$. Since $\xrightarrow{P} \subseteq \xrightarrow{P\#}$, by definition of $[\precsim\!\!\!\lhd]_{P\#}$ there exists $J \xleftarrow{P\#}\!\!\!\!\leftarrow \mathfrak{C}[N]$ with $\omega(M') \precsim\!\!\!\lhd\, \omega(J)$. Then we may take $N' = J$, since $\xrightarrow{P\#}\!\!\!\!\rightarrow$ and $\precsim\!\!\!\lhd$ conservatively extend $\xrightarrow{P}\!\!\!\!\rightarrow$ and $\leqslant$, respectively.

Proof: Lemma 6.1

Assume $H \xrightarrow{AR}\!\!\!\!\rightarrow J$. Let $\mathfrak{C} \in \mathrm{Ctx}(\mathcal{S}^{\#})$ be a closing context for H and J. Suppose that $H' \xleftarrow{P\#}\!\!\!\!\leftarrow \mathfrak{C}[H]$. We must show that there exists $J' \xleftarrow{P\#}\!\!\!\!\leftarrow \mathfrak{C}[J]$ such that $\omega(H') \precsim\!\!\!\lhd\, \omega(J')$. Compatibility of $\xrightarrow{AR}\!\!\!\!\rightarrow$ follows from compatibility of the R_i^* (by induction on the length of the $\xrightarrow{AR}\!\!\!\!\rightarrow$ sequence), thus, by our initial assumption we have $\mathfrak{C}[H] \xrightarrow{AR}\!\!\!\!\rightarrow \mathfrak{C}[J]$. Then, by commutativity, we have $H' \xrightarrow{AR}\!\!\!\!\rightarrow J' \xleftarrow{P\#}\!\!\!\!\leftarrow \mathfrak{C}[J]$ for some J'. Since each R_i satisfies (1), so does $\xrightarrow{AR}\!\!\!\!\rightarrow$ (by induction on the length of the $\xrightarrow{AR}\!\!\!\!\rightarrow$ sequence) and hence $\omega(H') \precsim\!\!\!\lhd\, \omega(J')$.

Proof: Theorem 6.5

We begin by factoring $\precsim\!\!\!\lhd$ into the composition of a more primitive form of abstraction with $\lesssim$. We then prove commutativity using Vincent van Oostrom's method of decreasing diagrams [23].

Definition A.1 (Simple Abstraction). *Let H, J be abstract terms in canonical form such that $\mathrm{FV}(J) \subseteq \mathrm{FV}(H)$ and $\mathrm{BV}(J) \subseteq \mathrm{BV}(H)$. Define the relation $\leq\ \subseteq \mathrm{vert}(H) \times \mathrm{vert}(J)$ thus: $v \leq w$ iff $w = v \vee w = \top$. Then H* is simply abstracted by *J, written $H \lhd J$, if:*

1. $\mathrm{root}(H) \leq \mathrm{root}(J)$*;*
2. *if x is bound to $S^k(\boldsymbol{v})$ in H and $x \in \mathrm{BV}(J)$, then x is bound to $S^k(\boldsymbol{w})$ in J and $\forall i, 1 \leq i \leq k.\, v_i \leq w_i$.*

$\lhd$ is easily seen to be a partial order.

Lemma A.2. *1.* $\lhd \subseteq \precsim\!\!\!\lhd$.
2. $H \precsim\!\!\!\lhd K$ *iff there exists J such that* $H \lhd J \lesssim K$.

To use van Oostrom's method, we must present $\xrightarrow{P\#}$ and $\xrightarrow{AR}$ as unions of labelled subrelations (we actually present slightly different relations which have the same transitive reflexive closure as these). Let $\xrightarrow{\oplus L}$ be the reflexive closure of the rewrite relation generated by just the $\oplus$-lifting rules in $P\#$ and let $\xrightarrow{P\top}$ be the

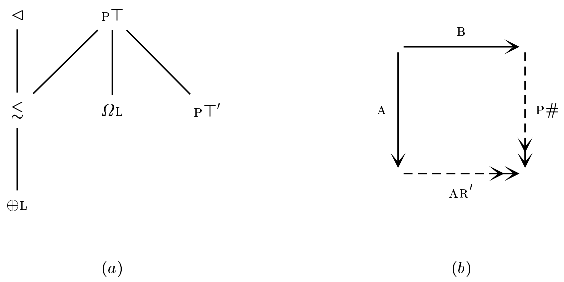


Fig. 1. applying van Oostrom's method

reflexive closure of the rewrite relation generated by $\textsc{p}$ plus the $\top$ rules in $\textsc{p}\#$. It is immediate from the definitions that

$$\xrightarrow{\textsc{p}\#}_{=} \;=\; \xrightarrow{\textsc{p}\top} \cup \xrightarrow{\oplus\textsc{l}}$$

Now let $\xrightarrow{\textsc{p}\top'} = \xrightarrow{\textsc{p}\top}$, let $\xrightarrow{\lhd} = \lhd$, let $\xrightarrow{\lesssim} = \lesssim$, and define:

$$\xrightarrow{\textsc{ar}'} = \xrightarrow{\textsc{p}\top'} \cup \xrightarrow{\oplus\textsc{l}} \cup \xrightarrow{\Omega\textsc{l}}_{=} \cup \xrightarrow{\lhd} \cup \xrightarrow{\lesssim}$$

It is an easy consequence of Lemma A.2 plus transitivity of $\mathrel{\underset{\sim}{\trianglelefteq}}$ that $\xrightarrow{\textsc{ar}'}\!\!\!\!\rightarrow = \xrightarrow{\textsc{ar}}\!\!\!\!\rightarrow$; thus it suffices to prove that $\xrightarrow{\textsc{ar}'}$ commutes with $\xrightarrow{\textsc{p}\#}$. Next we define a strict order $\prec$ on the labels of the subrelations; the Hasse diagram is shown in Fig. 1(a). Now we consider each $(\textsc{a}, \textsc{b}) \in \{\textsc{p}\top, \oplus\textsc{l}\} \times \{\textsc{p}\top', \oplus\textsc{l}, \Omega\textsc{l}, \lhd, \lesssim\}$, and show that, in each case, the diagram of Fig. 1(b) can be completed in a way which respects $\prec$ in the manner required by Theorem 2.3.5 of [23].

All the cases in $\{\textsc{p}\top, \oplus\textsc{l}\} \times \{\textsc{p}\top', \oplus\textsc{l}, \Omega\textsc{l}\}$ are very simple using subcommutativity (Prop. 2.9, Lemma 6.3(2)). The remaining four cases are summarised by the commuting squares in Fig. 2.

Cases (1), (2) and (3) are straightforward. Case (4) is more involved. In general, the $\textsc{p}\top$ redex rewritten down the left hand side will be just one member of a family comprising the set $\sigma^{-1}(\sigma(z))$, where z is the root of the $\textsc{p}\top$ redex and σ is the weak homomorphism witnessing $\lesssim$ along the upper edge. Intuitively, this family identifies a set of $\textsc{p}\top$ and $\Omega\textsc{l}$ redexes whose residuals we need to rewrite along the lower horizontal edge. However, not every member of this family need be the root of a redex, since the definition of weak homomorphism allows the presence of vertices labelled $\oplus$ to 'break' potential redexes. It can be shown that all these potential redexes can be exposed by use of $\lesssim$ and $\oplus\textsc{l}$. Similarly, $\sigma(z)$ may be the root of a 'broken' redex on the right, hence the use of $\oplus\textsc{l}$ rewrites down the right hand side. Formalising the argument involves the following key definition and lemmas:

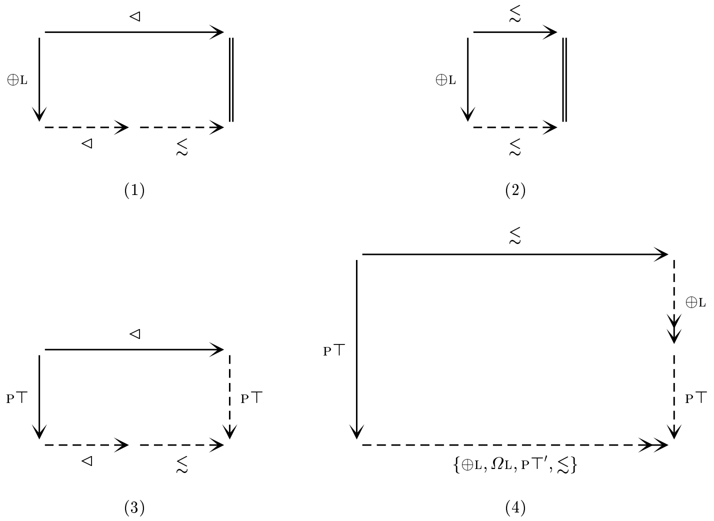


Fig. 2. proof cases for Theorem 6.5

Definition A.3 (Trace and Fragment). *Let H be in canonical form. Let Δ be a redex in H by a $\oplus$-lifting rule, let z be the root of Δ, and let $H \longrightarrow H'$ by Δ. By the definition of the $\oplus$-lifting rules (Def. 3.9), z is bound to $F(y_0, \boldsymbol{v})$ in H, for some pattern matching F, and y_0 is bound to $\oplus^k(\boldsymbol{w})$. As a result of the rewrite, all references to z in H are replaced by references to a fresh variable z' bound to $\oplus^k(z'_1, \ldots, z'_k)$ in H' with each z'_i bound to $F(w_i, \boldsymbol{v})[z'/z]$. We call z'* the trace of z by Δ *and we call the z'_i* the fragments of z by Δ.

Lemma A.4. *Say that $\sigma : H \lesssim K$ is* strong on z *if, for all $x \in \sigma^{-1}(z)$, if x is bound to $F(\boldsymbol{v})$ and F is pattern-matching, then v_1 is not labelled $\oplus$.*

Let K contain a redex Δ by one of the $\oplus$-lifting rules, let the root of Δ be z, and let $K \longrightarrow K'$ by Δ. If $\sigma : H \lesssim K$ is strong on z then there exists $\sigma' : H \lesssim K'$ such that σ' is strong on each of the fragments of z by Δ.

Lemma A.5. *Let $\Delta = (\rho, \sigma_\Delta)$ be a $\mathrm{P}\top$-redex in K and let z be the root of Δ. Let $\sigma : H \lesssim K$ be strong on z and let $\{x_1, \ldots, x_n\} = \{x \in \sigma^{-1}(z) \mid x$ is not bound to $\Omega\}$. Then:*

1. *There are only two possibilities for each x_i: (a) x_i is the root of a redex $\Delta_i = (\rho, \sigma_i)$ in H; (b) x_i is bound to $F(y_i, \boldsymbol{v})$ in H with F pattern-matching and y_i bound to Ω (thus x_i is the root of a $\Omega\mathrm{L}$ redex).*
2. *For $1 \le i \le n$ define J_i according to the cases in part (1): (a) $J_i = (\mathrm{rhs}(\rho))^{\sigma_i}$; (b) $J_i = \Omega$. If $K \longrightarrow K'$ by Δ, then $\mathrm{GC}(\overline{H[\boldsymbol{x} \leftarrow \boldsymbol{J}]}) \lesssim K'$.*