LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

# Graph Rewriting Systems

Computation Structures Group Memo 323-1

**Zena M. Ariola**
**Arvind**

This work was done at the Laboratory for Computer Science at MIT, and at INRIA. Funding for this work has been provided in part by the Advanced Research Projects Agency of the U.S. Department of Defense under the Office of Naval Research contracts N00014-84-K-0099 (MIT) and N0039-88-C-0163 (Harvard), and in part by ESPRIT contract SEMAGRAPH/BRA 3074 (INRIA).

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

# Graph Rewriting Systems

**Z.M. Ariola and Arvind**

## 2.1 Introduction

A modern trend in programming language theory has been to develop calculi to capture some specific aspects of functional language implementations. For example, several calculi for explicit substitution have recently been developed by Curien and Lévy [ACCL90, Cur86, Cur91, HL89, Hin77]. An attempt to formalize "weak reduction", *i.e.*, the kind of reduction that is actually done by most functional language implementations, is described by Maranget [Mar91]. Barendregt et al., have put forth a calculus to capture sharing in graph reduction implementation of term rewriting systems (TRSs) [BBvE$^+$87, BvEG$^+$87, Ken90, BvEvLP87]. In the same vein, we want to develop a calculus to capture the sharing of subexpressions in a more general class of languages.

Specification of sharing is desirable in the intermediate language used by a compiler for a purely functional language. Consider the function definition $\mathsf{F}\ x = x + x$ and the expression $\mathsf{F}(2+3)$. Any decent implementation, independently of the evaluation strategy (normal-order or applicative-order) it employs, will evaluate the subexpression $2+3$ only once. Dealing with sharing is important if the intermediate language is to be used to express and reason about optimizations. However, sharing becomes a necessity when a functional language is extended with side-effect operations, like I-structures [ANP89]. In general side-effects destroy "referential transparency" in the sense that the definition of an identifier cannot be substituted for each occurrence of the identifier in an unrestricted manner. Thus, the semantics of such a language requires a precise specification of sharing and substitution.

A way to capture sharing is to represent the expression as a graph instead of a linear text string or tree. This allows sharing of identical terms through pointers, and avoids repeated evaluation of identical terms as it is commonly done in normal-order reduction. Graph reduction for the $\lambda$-calculus was proposed by Wadsworth in order to bring together the advantages of both the applicative and the normal order

Term Graph Rewriting: Theory and Practice, eds. M.R.Sleep, M.J.Plasmeijer and M.C. van Eekelen

evaluation [Wad71]. Wadsworth also formally proved the correctness of his graph reduction technique. (As an aside - Wadsworth also showed that his graph reduction did not capture enough of sharing of expressions to lead to an optimal interpreter. More recently a new graph structure which allows sharing of "*contexts*", has been proposed by Kathail [Kat90]. This latter technique leads to provably optimal interpreters for the $\lambda$-calculus [L78]).

Much of the past work on graph rewriting has been to prove its correctness with respect to either the $\lambda$-calculus or Term Rewriting Systems (TRSs). We see graph rewriting as a system in its own right, and want to explore its syntactic and semantic properties. We want to include graphs with cycles and rewriting rules that recognize or create cycles. This is not the case in either [Wad71] or [BvEG+87] where only acyclic graphs are considered and thus, some important implementation ideas are ruled out.

In the following we formally introduce Graph Rewriting Systems (GRSs), and prove several syntactic properties of such systems. We also develop a term model for a restricted class of GRSs along the lines of Lévy's term model for $\lambda$-calculus. The restricted GRSs which we consider are adequate to describe sharing in combinatory systems but not the $\lambda$-calculus or the I-structures. In the last section we briefly discuss the applicability of our term model in showing the correctness of compiler optimizations.

This paper is based on the Ph.D. thesis of Zena M. Ariola [Ari92] where complete proofs and more examples with explanations may be found.

## 2.2 Syntax of GRSs

Our formalism for graph rewriting is based on the observation that a natural way to represent a graph textually is to associate an identifier to each node of the graph, and then write down all the interconnections as a *recursive let-block*. Equivalently we can say that we associate a name to each subexpression of a term. For example, the term $\mathsf{F}(+(2,2))$ will be expressed as:

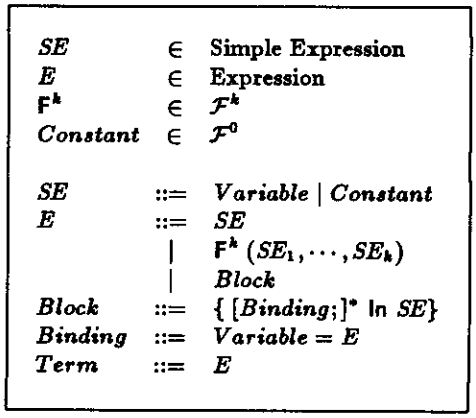
$$\begin{array}{l}\{\ t_1 = +(2,2);\\ \quad t_2 = \mathsf{F}(t_1)\\ \quad \mathsf{In}\ t_2\}\end{array}$$

In applying the rule $\mathsf{F}(x) \longrightarrow \mathsf{G}(x,x)$, the name $t_1$, and not the expression $+(2,2)$, will be substituted for each occurrence of $x$, leading to the term:

$$\begin{array}{l}\{\ t_1 = +(2,2);\\ \quad t_2 = \mathsf{G}(t_1,t_1)\\ \quad \mathsf{In}\ t_2\}\end{array}$$

We will allow the substitution of $+(2,2)$ for each free occurrence of $t_1$ only when $+(2,2)$ becomes a *value*, *i.e.*, 4. Thus, no duplication of work occurs during reduction. Therefore, we think that *an essential feature of a language for graph rewriting is the block construct with a suitable notion of substitutable values.*

The syntax of GRS terms is given in Figure 2.1. Superscript on a function symbol indicates its "arity" *i.e.*, the number of arguments it is supposed to have; constants



| | | |
|---|---|---|
| $SE$ | $\in$ | Simple Expression |
| $E$ | $\in$ | Expression |
| $\mathsf{F}^k$ | $\in$ | $\mathcal{F}^k$ |
| *Constant* | $\in$ | $\mathcal{F}^0$ |
| $SE$ | ::= | *Variable* \| *Constant* |
| $E$ | ::= | $SE$ |
| | \| | $\mathsf{F}^k\ (SE_1, \cdots, SE_k)$ |
| | \| | *Block* |
| *Block* | ::= | $\{\ [Binding;]^*\ \mathsf{In}\ SE\}$ |
| *Binding* | ::= | $Variable = E$ |
| *Term* | ::= | $E$ |

**Figure 2.1** Syntax of terms of a GRS with signature $\mathcal{F}$

are assumed to be function symbols of arity 0. The variable names on the left hand side of bindings in a block are required to be pairwise distinct. Furthermore the order of bindings in a block does not matter.

In the following $\mathtt{FV}(M)$ and $\mathtt{BV}(M)$ will denote the free and bound variables of term $M$, respectively. Moreover, if $M \equiv \{x_1 = e_1; \cdots x_n = e_n\ \mathsf{In}\ x\}$ and $x$ is a variable, we will say that $M$ is rooted at $x$. The root of a term plays a special role during its reduction. For example, consider the rule $\mathsf{F}(\mathsf{G}(x)) \longrightarrow 0$ and the term $\mathsf{F}(\mathsf{G}(1))$. During the reduction only the pointers to $\mathsf{F}$ (*i.e.*, the root) are redirected to 0, and the subterm $\mathsf{G}(1)$ remains unaffected. We call the subterm $\mathsf{G}(x)$ the *precondition* of the above rule, which will be written in the GRS notation as follows:

$$\frac{x_2 = \mathsf{G}(x)}{x_1 = \mathsf{F}(x_2) \longrightarrow x_1 = 0}$$

DEFINITION **2.2.1 (GRS rule)** *A GRS rule $\tau$ is a set of preconditions, $x_1 = e_1, \cdots, x_n = e_n$, and a left-hand-side, $l$, and a right-hand-side, $r$, and is written as:*

$$\frac{x_1 = e_1 \mid \cdots \mid x_n = e_n}{x = l \ \longrightarrow \ x = r}$$

*where*

*a. $e_i, 1 \leq i \leq n$, and $l$ are terms of the form $\mathsf{F}^k(y_1, \cdots, y_k)$, where each $y_i$ is either a variable or a constant and $k > 0$;*

*b. $r$ is a term such that $\mathtt{FV}(r) \subseteq \mathtt{FV}(\{x_1 = e_1; \cdots x_n = e_n;\ x = l\ \mathsf{In}\ x\}) \cup \{x_1, \cdots, x_n, x\}$.*

*The term $\{x_1 = e_1; \cdots x_n = e_n;\ x = l\ \mathsf{In}\ x\}$ is called the pattern of rule $\tau$, and is denoted by $\mathcal{P}(\tau)$.*

The metavariables of a rule correspond to the free variables of its pattern. Notice that restriction (*a*) makes it impossible to give a GRS rule to rewrite a constant

or a variable. However, we do not restrict the pattern of a rule to be a strongly connected rooted graph. Such a rule, referred to as a *multi-rooted rule*, can be used to describe side-effect operations by keeping the state of the store directly in the term. For example, we can express the operation of reading location $I$ of array $X$ as follows:

$$\frac{x_1 = \mathsf{Store}(X, I, Z)}{x = \mathsf{Select}(X, I) \longrightarrow x = Z}$$

which can be assigned a natural interpretation that if there is a Store in the "context" of a Select then the Select can be rewritten to $Z$. It is for this reason that in our earlier work we had called our system a *Contextual Rewriting System* [AA89, AA91a, AA91b]. However, Jean-Jacques Lévy convinced us that our system basically described graph rewriting, so we renamed it simply a GRS.

DEFINITION **2.2.2 (GRS)** *A GRS is a structure $(A(\mathcal{F}), R)$, where $A(\mathcal{F})$ is the set of GRS terms defined over signature $\mathcal{F}$, and $R$ is a set of GRS rules.*

## 2.3 Basic Rules of GRSs

Analogous to the notion of $\alpha$-equivalence in $\lambda$-calculus, we want to identify GRS terms whose differences may be regarded as merely syntactic noise. To that end we assume that all GRSs come equipped with some basic set of "rules". These rules fall outside the syntax prescribed in Definition 2.2.1.

**Substitution rules:**

$$\frac{\mathsf{X} = \mathsf{V}}{\mathsf{X} \longrightarrow \mathsf{V}} \qquad \frac{\mathsf{X} = \mathsf{Y}}{\mathsf{X} \longrightarrow \mathsf{Y}} \quad \mathsf{X} \not\equiv \mathsf{Y}$$

where $V$ is a constant. These rules formalize the notion of a substitutable expression and say that only constants and variables (provided $X$ and $Y$ are distinct variables) can be substituted freely. The corresponding binding can be deleted from the term when all such substitutions have been performed.

**Degenerate cycle rule:**

$$\mathsf{X} = \mathsf{X} \longrightarrow \mathsf{X} = \mathsf{o}$$

This rule says that if we encounter a nonsensical binding like $x = x$ then we bind $x$ to the special symbol o. The special symbol o behaves just like a constant value and can be substituted freely.

**Block Flattening rule:**

$$\begin{array}{l} \{y = \{\, SS_1;\ SS_2;\ \cdots \\ \qquad\qquad \mathsf{In}\ x\} \\ \quad S_1;\ \cdots\ S_n \\ \qquad \mathsf{In}\ z\} \end{array} \longrightarrow \begin{array}{l} \{y = x' \\ \quad SS_1'\ SS_2' \cdots \\ \quad S_1;\ \cdots\ S_n \\ \qquad \mathsf{In}\ z\} \end{array}$$

where $x'$ and $SS_i'$ indicate renaming of all bound variables occurring in the internal

block to avoid name clashes with the names in the surrounding scope.

**Commutativity rule:**

$$\{\cdots S_i;\ S_j \cdots \mathsf{In}\ x\} \longrightarrow \{\cdots S_j;\ S_i \cdots \mathsf{In}\ x\}$$

This rule says that the order of bindings in a block does not affect a term.

A term is said to be in *canonical form* if all the substitutions, the detection of degenerate cycles, and the flattening of blocks have been performed, and all bindings of the form $x = y$ and $x = v$ have been deleted. Consequently, two terms $M$ and $N$ are said to be *$\alpha$-equivalent* if their canonical forms are the same up to renaming of bound variables and commutativity of bindings.

## 2.4 Identifying Redexes and Reduction

There are subtle issues involved in identifying redexes in a term. Consider the following two rules:

$$\tau_1 : \frac{x_1 = \mathsf{F}(0) \mid x_2 = \mathsf{F}(0)}{x = \mathsf{G}(x_1, x_2) \longrightarrow x = 0} \qquad \tau_2 : \frac{x_1 = \mathsf{F}(0)}{x = \mathsf{G}(x_1, x_1) \longrightarrow x = 0}$$

and the following two terms:

$$M \equiv \{\ t_1 = \mathsf{F}(0);\ t_2 = \mathsf{F}(0);\ t_3 = \mathsf{G}(t_1, t_2)\ \mathsf{In}\ t_3\} \qquad N \equiv \{\ t_1 = \mathsf{F}(0);\ t_2 = \mathsf{G}(t_1, t_1)\ \mathsf{In}\ t_2\}$$

Intuitively we can say that $\tau_1$ matches $M$ with the substitution "$x = t_3, x_1 = t_1, x_2 = t_2$", and $\tau_2$ matches $N$ with substitution "$x = t_2, x_1 = t_1$". Does rule $\tau_1$ apply to $N$? Or does rule $\tau_2$ apply to $M$? Rule $\tau_1$ does indeed apply to the term $N$ by matching both the preconditions by the same binding, that is, by considering the substitution "$x_1 = t_1, x_2 = t_1, x = t_2$". However, there will not be any variable substitution that makes $\tau_2$ applicable to $M$. Thus, the preconditions of a rule can be satisfied by overlapping bindings. Moreover, the lhs of a rule can also overlap its precondition, as shown in the following example. Consider the term $M \equiv \{t = \mathsf{G}(t)\ \mathsf{In}\ t\}$ and the rule:

$$\frac{x_1 = \mathsf{G}(Y)}{x = \mathsf{G}(x_1) \longrightarrow x = 0}$$

The substitution "$x = t, x_1 = t, Y = t$" makes $\mathsf{G}(t)$ both a redex and its precondition!

We can capture the notion of a redex in terms of an ordering on terms. For this purpose we extend the syntax of GRS terms with a new "constant", called $\Omega$, which matches any term and is less than or equal to any term in our ordering. The constant $\Omega$, however, behaves differently than other constants because *$\Omega$ is not a substitutable value*. Thus, the term $\{t_1 = \Omega;\ t_2 = \Omega;\ t_3 = \mathsf{G}(t_1, t_2)\ \mathsf{In}\ t_3\}$ is not the same as $\{t_1 = \Omega;\ t_2 = \mathsf{G}(t_1, t_1)\ \mathsf{In}\ t_2\}$.

DEFINITION **2.4.1** (**$\omega$-ordering: $\leq_\omega$**) *Given GRS terms $M$ and $N$ in canonical form, rooted at $z_1$ and $z_2$, respectively, $M \leq_\omega N$ iff $\exists$ a function $\sigma : (\mathtt{BV}(M) \cup \mathtt{FV}(M) \cup \mathcal{F}^0) \rightarrow (\mathtt{BV}(N) \cup \mathtt{FV}(M) \cup \mathcal{F}^0)$ such that:*

*a.* $\forall c \in \mathcal{F}^0$, $\sigma(c) = c$;
*b.* $\forall x \in \mathtt{FV}(M)$, $\sigma(x) = x$;
*c.* $\forall x \in \mathtt{BV}(M)$, *if $x$ is bound to* $\mathsf{F}^k(y_1, \cdots, y_k)$ *in $M$ then* $\exists z, z = \sigma(x)$ *such that $z$ is bound to* $\mathsf{F}^k(\sigma(y_1), \cdots, \sigma(y_k))$ *in $N$;*
*d.* $\sigma(z_1) = z_2$.

*The function $\sigma$ is called the induced substitution.*

Notice that if a variable is bound to an $\Omega$, condition $(c)$ is automatically satisfied. Intuitively, $M \leq_\omega N$ if $N$ can be obtained from $M$ by replacing $\Omega$ with any other term or by increasing the sharing in $M$. Thus, $M \leq_\omega N$, where $M$ and $N$ are the terms in the example given at the beginning of this section.

We use $\omega$-ordering as follows in defining a redex. We substitute $\Omega$ for all metavariables in the pattern of a rule. Such a term is called the *closure of a rule*. If term $p$ is the closure of rule $\tau$ then a term $M$ is said to be a $\tau$-redex if $p \leq_\omega M$.

DEFINITION **2.4.2** (**Closure of a Rule**) *Given a GRS rule $\tau$, the closure of $\tau$, written as $Cl(\tau)$, is the term $\{y_1 = \Omega; \cdots y_m = \Omega;\ t = \mathcal{P}(\tau)\ \mathsf{In}\ t\}$, where $\{y_1, \cdots, y_m\} = \mathtt{FV}(\mathcal{P}(\tau))$ and $t$ is a new variable.*

In the following, we will make use of the notation $M@x_i$, where $x_i \in \mathtt{BV}(M)$, which stands for the term $M$ rooted at $x_i$, that is, the term $M@x_i$ is the same as $M$ except that it is rooted at $x_i$. For example, if $M \equiv \{x_1 = e_1; \cdots x_n = e_n\ \mathsf{In}\ x\}$ then $M@x_i$ will be the term $\{x_1 = e_1; \cdots x_n = e_n\ \mathsf{In}\ x_i\}$.

DEFINITION **2.4.3** (**Redex**) *A redex in a GRS term $M$ is a triple $(\tau, z, \sigma)$ such that*

*a.* *$\tau$ is a GRS rule;*
*b.* $z \in \mathtt{BV}(M)$;
*c.* *$Cl(\tau) \leq_\omega M@z$ and $\sigma$ is the induced substitution.*

DEFINITION **2.4.4** (**Instance of a Term**) *Given a GRS term $M$ and a substitution $\sigma$, an instance of $M$, written as $M^\sigma$, is the term obtained by substituting $\sigma(x)$ for each free variable $x$ of $M$ and renaming each bound variable of $M$.*

Given a rule $\tau : \dfrac{x_1 = e_1 \mid \cdots \mid x_n = e_n}{x = l \rightarrow x = r}$, and redex $(\tau, z, \sigma)$ occurring in $M$, the reduction step consists of first allocating $r^\sigma$, that is, an instance of the rhs of rule $\tau$ using substitution $\sigma$. Subsequently, the term bound to the root of the redex is replaced by the newly instantiated term. The replacement operation is written as $M[z \leftarrow r^\sigma]$. The term so obtained is then *canonicalized.*

DEFINITION **2.4.5** (**Reduction, $\longrightarrow$**) *Given a GRS term $M$ in canonical form and rule $\tau : \dfrac{x_1 = e_1 \mid \cdots \mid x_n = e_n}{x = l \rightarrow x = r}$, $M$ reduces to $N$ by doing the $\tau$-redex at $z$ in $M$ (written as $M \xrightarrow[z]{\tau} N$), iff $(\tau, z, \sigma)$ is a redex in $M$ and $N \equiv_\alpha M[z \leftarrow r^\sigma]$.*

The instantiation of $r$ in our system corresponds to the *build phase* of Barendregt system while our replacement operation corresponds to his *redirection phase*. However, there are subtle differences between the two systems. First of all, the *garbage collection phase* of Barendregt (that is, the deletion of nodes that are not reachable from the root), which in our system will correspond to the *dead code elimination*, is not performed in our GRS. The rational being that we want to allow the so called multi-rooted rules. The other difference arises in the presence of "projection" rules and cyclic graphs. For example, given the rule $x = \mathsf{I}(X) \longrightarrow x = X$, and the cyclic term $M \equiv \{t = \mathsf{I}(t)\ \mathsf{In}\ t\}$, following the Barendregt system, we will have, $M \longrightarrow M$. According to our system, $M \longrightarrow \{t = t\ \mathsf{In}\ t\}$, which, as explained before, will become o, a symbol to represent a "meaningless" term. This difference has a strong impact on the confluence of GRSs, as we will see shortly.

## 2.5 Confluent GRSs

Not all GRSs are confluent, however, we can show that for a restricted class, namely GRSs without interfering rules, confluence is guaranteed. We introduce the notion of *compatible terms* which will be used, among other things, to define the notion of interference among rules. The idea is that terms which are not ordered may still have a common upper bound. As we will see such terms can potentially interfere with each other.

DEFINITION **2.5.1 (Compatible terms, $\uparrow_\omega$)** *Given GRS terms $M_1$ and $M_2$ in canonical form, $M_1$ and $M_2$ are said to be $\omega$-compatible, written as $M_1 \uparrow_\omega M_2$, iff $\exists M_3$ such that $M_1 \leq_\omega M_3$ and $M_2 \leq_\omega M_3$.*

For example, the term $\{t_1 = \Omega;\ t_2 = \Omega;\ t = \mathsf{G}(t_1, t_2)\ \mathsf{In}\ t\}$ is compatible with the term $\{t_1 = \mathsf{F}(0);\ t_2 = \mathsf{F}(0);\ t = \mathsf{G}(t_1, t_2)\ \mathsf{In}\ t\}$ but not with the term $\{t_1 = \mathsf{F}(0);\ t_2 = \mathsf{H}(0);\ t = \mathsf{G}(t_1, t_2)\ \mathsf{In}\ t\}$.

DEFINITION **2.5.2 (Interference)** *Given GRS rules $\tau_1$ and $\tau_2$, $\tau_1$ is said to interfere with $\tau_2$ iff $\exists x \in \mathsf{BV}(\mathcal{P}(\tau_1))$ such that*

*a. if $\tau_1 \neq \tau_2$ then $Cl(\tau_1)@x \uparrow_\omega Cl(\tau_2)$;*

*b. if $\tau_1 = \tau_2$ and $\tau_1$ is a single-rooted rule then $Cl(\tau_1)@x \uparrow_\omega Cl(\tau_2)$, where $x$ is not the root of $\mathcal{P}(\tau_1)$.*

For example, the rule $\tau$ : $\dfrac{x_1 = \mathsf{L}(Y)}{x = \mathsf{L}(x_1) \longrightarrow x = 0}$ will interfere with itself because $Cl(\tau)@x_1 \uparrow_\omega Cl(\tau)$. Notice that the following two rules are non-interfering,

$$\frac{x_1 = \mathsf{G}(Y)}{x = \mathsf{F}(x_1) \longrightarrow x = 0} \qquad \frac{x_1 = \mathsf{G}(Y)}{x = \mathsf{D}(x_1) \longrightarrow x = 1}$$

It can be seen from this example that the preconditions of non-interfering rules are not affected by a reduction. We also note in passing that multi-rooted rules are always self-interfering, because they may cause an overlapping at the root.

In the following we will write a GRS with non-interfering rules as $\text{GRS}_{\mathsf{NI}}$.

THEOREM 2.5.3 *Given a* $GRS_{\mathsf{NI}}$ *term* $M$, *if* $M \underset{x_1}{\longrightarrow} M_1$ *and* $M \underset{x_2}{\longrightarrow} M_2$ *then* $\exists\, M_3$ *such that* $M_2 \underset{x_1}{\longrightarrow} M_3$ *and* $M_1 \underset{x_2}{\longrightarrow} M_3$.

COROLLARY 2.5.4 *A* $GRS_{\mathsf{NI}}$ *is confluent up to* $\alpha$*-equivalence.*

Consider the projection rules, $x = \mathsf{I}(X) \longrightarrow x = X$ and $x = \mathsf{J}(X) \longrightarrow x = X$ and the term $M \equiv \{x = \mathsf{I}(y);\ y = \mathsf{J}(z)\ \mathsf{In}\ x\}$, then $M \longrightarrow M_1 \equiv \{x = \mathsf{I}(z)\ \mathsf{In}\ x\}$ and $M \longrightarrow M_2 \equiv \{x = \mathsf{J}(z)\ \mathsf{In}\ x\}$. Notice that if both $M_1$ and $M_2$ are not reduced to $\mathsf{o}$, the confluence property will be lost, as was observed in [KKSdV91]. Barendregt's graph reduction system is not confluent precisely because of the absence of such a reduction.

## 2.6 A Graph Model for GRSs

We are interested in defining an equality on the set of terms such that the equality is useful in analyzing the correctness of compiler optimizations. Thus, we have to guarantee that if two terms $M$ and $N$ are equal, then the equality is preserved by putting them in the same context, *i.e.*, $M = N \Longrightarrow \forall C[\Box], C[M] = C[N]$. This means that the equality has to be a *congruence* with respect to the formation rules of terms. Only then equal terms will be substitutable for each other, and, an optimization will be considered correct if it preserves equality.

An example of an equivalence relation on terms is *convertibility*. However, convertibility is too restrictive from a compiler's point of view, as shown by the following example:

$$M \equiv \{\ x = \mathsf{Cons}(y, z);\ y = \mathsf{F}(0);\ z = \mathsf{Cons}(y, \mathsf{Nil})\ \mathsf{In}\ x\}$$

$$N \equiv \{\ x = \mathsf{Cons}(y, z);\ y = \mathsf{F}(0);\ z = \mathsf{Cons}(w, \mathsf{Nil});\ w = \mathsf{F}(0)\ \mathsf{In}\ x\}$$

where $M$ and $N$ are in normal forms but not convertible to each other. However, if the internal representation of lists is ignored by an observer then both the terms represent the same *unfolded* list, $\mathsf{F}(0) : \mathsf{F}(0) : \mathsf{Nil}$. If the GRS containing these terms has a non left-linear rule, it may be possible to distinguish between such terms. Thus, such terms cannot be equated without disallowing non left-linear rules.

We should also notice that $N \leq_\omega M$, *i.e.*, $N$ has "less sharing" than $M$ in the above example. Does it mean that $N$ is "less defined" than $M$ in the sense that one can *compute less* with $N$ than with $M$? We would like to answer this question without delving into heavy duty model theory. We have carefully said "compute" to emphasize that we are interested in studying what a term represents from an operational point of view. In particular, we are interested in observing the gradual *syntactic building up of the final term.*

We introduce a function $\omega$ to compute the stable part of a term, that is, the part of the term that will not change as more reductions are performed on it. The $\omega$ function

captures what Lévy has called the *direct approximation* of a $\lambda$-calculus term [Lé78], and Welch has called the *instantaneous semantics* of a term [Wel75]. Notice that as more reductions are performed the stable part should get larger, that is, if $M \longrightarrow M_1 \longrightarrow M_2 \cdots$ then $\omega(M) \leq_\omega \omega(M_1) \leq_\omega \omega(M_2) \cdots$. We remind the reader that $\leq_\omega$ is the syntactic ordering on terms that captures both the sharing, and the fact that $\Omega$ is less than any other term.

We collect all the stable or observable information gathered by reducing $M$ in a set, called $W^*(M)$, and say that it represents the *information content* of $M$. We can now formulate our original question regarding the impact of sharing on a program's behavior as follows: *if $N$ has less sharing than $M$ then is $W^*(N)$ contained in $W^*(M)$?* As we shall see shortly, this is indeed the case in the absence of interfering rules.

It is also interesting to analyze if $N$ is "less defined" than $M$ implies that for all context $C[\Box]$, $C[N]$ is "less defined" than $C[M]$. That is, is the equality induced by $W^*$ a congruence? We will see that in the absence of interfering rules, the equality is also a congruence. Thus, we can conclude that the collection of stable information contained in GRS terms is indeed a model for GRSs without interfering rules.

### 2.6.1 Instant Semantics

The instant semantics of a GRS term $M$ consists of computing its *stable part*, where by stable part we mean the part of $M$ which will not change by further reductions. The first intuitive solution that comes to mind is to replace all redexes in a term by $\Omega$ (since a redex sub-expression can become any expression and $\Omega$ is less than all expressions). This solution has a problem as shown by the following example. Consider the rules:

$$\tau_1 : x = \mathsf{F}(Y,Y) \longrightarrow x = \gamma \qquad \tau_2 : x = \mathsf{I}(Y) \longrightarrow x = Y$$

and the following reduction:

$$M \equiv \{\ t = \mathsf{F}(t_1,t_2);\ t_1 = \mathsf{A}(0);\ t_2 = \mathsf{I}(t_1)\ \mathsf{In}\ t\} \xrightarrow[t_2]{\tau_2} M_1 \equiv \{\ t = \mathsf{F}(t_1,t_1);\ t_1 = \mathsf{A}(0)\ \mathsf{In}\ t\}$$

The only redex in $M$ is rooted at $t_2$. Suppose we replace it by $\Omega$ to obtain the stable term $M_2 \equiv \{\ t = \mathsf{F}(t_1,t_2);\ t_1 = \mathsf{A}(0);\ t_2 = \Omega\ \mathsf{In}\ t\}$. However, since the root of $M_1$ is a redex the stable information in $M_1$ is less than the stable information in $M$. This is contrary to our intuition that the information should increase with reduction. The problem is due to the presence of rule $\tau_2$ which can introduce sharing. If we want to compute the instant semantics of a term without analyzing the rhs of rules then we have to assume that the $\Omega$ in $M_2$ can be replaced by the node with label A, and thus, can make the node with label F a $\tau_1$-redex. Therefore we should not treat node F in $M_2$ as stable information. This example shows clearly that the first solution does not work. However, it does work for recursive program schema (RPS)!

The problem that the above example illustrates is that, even though $M$ is not a redex it can become a redex when some redexes under it are performed. This phenomena is usually called *upward creation of redexes*. Reduction of a term in the $\lambda$-calculus or TRSs can also result in the upward creation of redexes. However, upward cre-

ation of redexes is not possible in RPSs. To cope with this problem in the $\lambda$-calculus, Wadsworth [Wad71] and Lévy [Lé78] have introduced the notion of the $\omega$-rule, which states:

$$\Omega\, M \longrightarrow \Omega$$

This simple $\omega$-rule reduces any term that can become a redex (by upward creation) to $\Omega$. However, the presence of non left-linear rules makes the generation of $\omega$-rules for TRSs and GRSs difficult. Therefore, instead of introducing $\omega$-rules, we introduce the notion of a *compatible redex* or an *$\omega$-redex*. A compatible redex captures our intuition about why a term should be rewritten to $\Omega$. It consists of analyzing a term to see if it can become a redex either by replacing $\Omega$ with some other term or by increasing the sharing in the term.

**Definition 2.6.1 (Compatible Redex)** *A compatible redex in a GRS term $M$ is a pair $(\tau, z)$ such that:*

*a. $\tau$ is a rule;*
*b. $z \in \mathbf{BV}(M)$ and $z$ is not bound to $\Omega$;*
*c. $Cl(\tau) \uparrow_\omega M@z$ and $Cl(\tau) \not\leq_\omega M@z$.*

*$z$ is called the root of the compatible redex.*

Notice that because of condition $(c)$, a compatible redex cannot be an ordinary redex. For the example given at the beginning of this section, we have that $Cl(\tau_1) \not\leq_\omega M_2$ and $Cl(\tau_1) \uparrow_\omega M_2$, thus, $M_2$ is a compatible redex and as such should be reduced to $\Omega$.

**Definition 2.6.2 ($\omega$-reduction, $\underset{\omega}{\longrightarrow}$)** *Given GRS terms $M$ and $N$, $M$ $\omega$-reduces to $N$ by doing the $\tau$-compatible redex at $z$ (written as $M \underset{z\ \omega}{\overset{\tau}{\longrightarrow}} N$) iff $(\tau, z)$ is a compatible redex in $M$, and $N \equiv_\alpha M[z \leftarrow \Omega]$.*

*A GRS term $M$ is said to be in $\omega$-normal form if it does not contain any compatible redexes.*

**Proposition 2.6.3** *$\underset{\omega}{\longrightarrow\!\!\!\!\!\rightarrow}$ is confluent and strongly normalizing.*

The stable part of a term $M$, *i.e.*, $\omega(M)$, will then be computed by first replacing all distinct redexes occurring in $M$ by $\Omega$ and then computing the $\omega$-normal form of the term so obtained.

**Definition 2.6.4** *Given a GRS term $M$, $M_\Omega$ is the term $M[u_1 \leftarrow \Omega] \cdots [u_n \leftarrow \Omega]$ where $u_1 \cdots u_n$ are all the distinct redexes occurring in $M$.*

**Definition 2.6.5 ($\omega$-function)** *Given a GRS term $M$, $\omega(M)$ is the $\omega$-normal form of $M_\Omega$.*

### 2.6.2 Meaning of a GRS term

We collect all observable information about GRS terms in a set called $\omega$-graphs.

**DEFINITION 2.6.6 ($\omega$-graphs: Set of observations)** *Given a GRS, the set of all observations, called $\omega$-graphs, is defined as:*

$$\omega\text{-}graphs = \bigcup\{\omega(M) \mid \forall \ GRS\ terms\ M\}.$$

**DEFINITION 2.6.7 ($W^*$: The information content of a GRS term)** *Given a GRS term $M$, $W^*(M) = \{a \mid a \in \omega\text{-}graphs, a \leq_\omega \omega(M'), M \longrightarrow\!\!\!\!\!\rightarrow M'\}$.*

We have chosen to represent $W^*$ by a set as opposed to the least upper bound of the set for technical reasons.

**DEFINITION 2.6.8 ($\sqsubseteq_{\mathbf{g}}$: Information ordering)** *Given GRS terms $M$ and $N$, $M \sqsubseteq_{\mathbf{g}} N$ iff $W^*(M) \subseteq W^*(N)$.*

If we want $W^*$ to be our interpretation function $W^*$ will have to satisfy some properties, that is, the meaning will have to be preserved by reduction, and it will have to be compositional. In other words:

$$\begin{array}{lrcl} \textbf{Soundness}: & M \longrightarrow\!\!\!\!\!\rightarrow N & \Longrightarrow & M \equiv_{\mathbf{g}} N \\ \textbf{Congruence}: & M \equiv_{\mathbf{g}} N & \Longrightarrow & C[M] \equiv_{\mathbf{g}} C[N] \end{array}$$

In order to show soundness we need to show some additional properties of the $\omega$-function. In particular, we want to guarantee that the $\omega$-function is monotonic with respect to $\leq_\omega$. From this it will follow that the $\omega$-function is monotonic with respect to reduction.

**THEOREM 2.6.9 (Soundness of $\equiv_{\mathbf{g}}$)** *Given a confluent GRS and terms $M$ and $N$, if $M \longrightarrow\!\!\!\!\!\rightarrow N$ then $M \equiv_{\mathbf{g}} N$.*

### 2.6.3 Impact of Sharing on a Program Behavior

Before dealing with the question of congruence, let us digress and analyze the impact of sharing on a program behavior, that is, $M \leq_\omega N \Longrightarrow M \sqsubseteq_{\mathbf{g}} N$ ? It turns out that in the presence of interfering rules the above will not hold. Consider the following GRS which has *confluent* but interfering rules:

$$\begin{array}{ll} \tau_1: & \dfrac{x_1 = \mathsf{B}(0) \mid x_2 = \mathsf{C}(0)}{x = \mathsf{A}(x_1, x_2) \longrightarrow x = 0} \\ \tau_2: & x = \mathsf{C}(0) \longrightarrow x = \mathsf{C}(0) \\ \tau_3: & x = \mathsf{B}(0) \longrightarrow x = \mathsf{C}(0) \end{array}$$

and the following terms:

$$M \equiv \{\ t_1 = \mathsf{A}(t_2, t_3);\ t_2 = \mathsf{B}(0);\ t_3 = \mathsf{B}(0)\ \mathsf{In}\ t_1\}$$

$$N \equiv \{\ t_1 = \mathsf{A}(t_2, t_2);\ t_2 = \mathsf{B}(0)\ \mathsf{In}\ t_1\}$$

It is easy to see that $W^*(M) = \{\Omega, 0\}$, while $W^*(N) = \{\Omega, \{x_1 = \Omega;\ x_2 = \mathsf{A}(x_1, x_1)\ \mathsf{In}\ x_2\}\}$. Therefore, $M \leq_\omega N$ and $M \not\sqsubseteq_{\mathbf{g}} N$.

**THEOREM 2.6.10 (Monotonicity of $\sqsubseteq_{\mathbf{g}}$ with respect to $\leq_\omega$)** *Given $GRS_{\mathsf{NI}}$ terms $M$ and $N$, if $M \leq_\omega N$ then $M \sqsubseteq_{\mathbf{g}} N$.*

### 2.6.4 Congruence

We want to show that $\equiv_{\mathbf{g}}$ is a congruence, that is, if $M \equiv_{\mathbf{g}} N$ then $\forall C[\Box], C[M] \equiv_{\mathbf{g}} C[N]$. What can prevent $\equiv_{\mathbf{g}}$ from being a congruence? In the definition of observable behavior we may have discarded something important, which may have an effect on the context enclosing the term. Suppose we have chosen to observe booleans only, that is, we will distinguish between True and False, but not between 5 and 7. Thus, the two program $M \equiv 5$ and $N \equiv 7$ will trivially exhibit the same behavior. However, by putting them in the context $\{\ x = \Box;\ p =\leq (x, 5);\ y = \mathsf{Cond}(p, \mathsf{True}, \mathsf{False})\ \mathsf{In}\ y\}$ we will observe True when running $C[M]$ and False when running $C[N]$. It seems that we cannot discard any information that can be used to build terms.

A way of assuring that $\equiv_{\mathbf{g}}$ is a congruence is to show that for any context $C[\Box]$, the behavior of $C[M]$ can be inferred from the observations about $M$, that is,

$$\forall C[\Box], W^*(C[M]) = \bigcup\{W^*(C[P]) \mid P \in W^*(M)\}$$

In other words the context operation must be a continuous operation with respect to the observations. The proof that

$$\bigcup\{W^*(C[P]) \mid P \in W^*(M)\} \subseteq W^*(C[M])$$

follows automatically from the monotonicity of $\sqsubseteq_{\mathbf{g}}$ with respect to $\leq_\omega$. The other direction requires some more machinery.

We need to show that each observation of $C[M]$ can be obtained by plugging some observation of $M$, instead of $M$ itself, in the context $C[\Box]$. There are two basic steps in the proof:

(i) Suppose $C[M] \longrightarrow\!\!\!\!\!\rightarrow N$. Let $\mathcal{F}_1$ be the set of redexes in $M$ that must be reduced to get to $N$, and let $\mathcal{F}_2$ be the set of all other redexes in $M$. The first step of the proof consists of showing that each reduction can be reordered such that we first reduce all the redexes in $\mathcal{F}_1$, that is, $\exists M'$, $C[M] \underset{\mathcal{F}_1}{\longrightarrow\!\!\!\!\!\rightarrow} C[M']$. Let $\mathcal{F}_3$ be the set of redexes in $M'$ that are descendants of redexes in $\mathcal{F}_2$. We need to show that the rest of the reduction can be performed without reducing any redex in $\mathcal{F}_3$. Notationally we will say $C[M'] \overset{\not\mathcal{F}_3}{\longrightarrow\!\!\!\!\!\rightarrow} N$.

(ii) Next we need to show that the $\omega$-function does not lose too much information. Let $M_2$ be the term obtained from $M$ by setting all the redexes in $\mathcal{F}_2$ to $\Omega$. We first prove that if $C[M] \overset{\not\mathcal{F}_2}{\longrightarrow\!\!\!\!\!\rightarrow} N$, then the same reduction can be carried out on $C[M_2]$. Since W* contains terms in $\omega$-normal form, we also need to show that $\omega$-reductions do not destroy the meaning of a term.

**THEOREM 2.6.11 (Congruence of $\equiv_{\mathbf{g}}$)** *Given $GRS_{\mathsf{NI}}$ terms $M$ and $N$, if $M \equiv_{\mathbf{g}} N$ then $\forall C[\Box], C[M] \equiv_{\mathbf{g}} C[N]$.*

In the presence of interfering rules $\equiv_{\mathbf{g}}$ is not guaranteed to be a congruence. As an example consider the following rules:

$$r_1: \quad x = \mathsf{B}(Y) \longrightarrow x = \mathsf{B}(Y)$$
$$r_2: \quad x = \mathsf{A}(Y) \longrightarrow x = \mathsf{A}(Y)$$
$$r_3: \quad \frac{x_1 = \mathsf{A}(Y)}{x = \mathsf{F}(x_1) \longrightarrow x = 1}$$

then $\mathsf{A}(0) \equiv_{\mathbf{g}} \mathsf{B}(0)$, however, $\mathsf{F}(\mathsf{A}(0)) \not\equiv_{\mathbf{g}} \mathsf{F}(\mathsf{B}(0))$.

## 2.7 Conclusion

The motivation for this work came from a desire to formalize the compilation process of Id as a series of translations into simpler and simpler languages. To that end we have introduced the Kid (Kernel id) language [AA91a] and the P-TAC (Parallel Three Address Code) language [AA89]. We also provided the translation of Id into Kid and of Kid into P-TAC [AA91b]. P-TAC can be seen as an example of GRS, while Kid is more general due to the presence of $\lambda$-abstraction. This approach has lead to the formalization of compiler optimizations in terms of source-to-source transformations on these intermediate languages. Moreover, using the notion of information content of a term we have given a criteria for the (partial) correctness of these compiler optimizations [AA91a].

The results presented in this Chapter (notably theorems 2.6.10 and 2.6.11) can be applied in a straightforward manner to show the partial correctness of those optimizations that simply increase the sharing in a term. Examples of such optimizations include the common subexpression elimination, and the lifting of free expressions and loop invariants. We can also show the partial correctness of the cyclic Y-rule. In order to prove total correctness we need to discard sharing from our observations at the expenses of introducing more restrictions on the rules, as discussed in [Ari92].

We would like to extend GRSs with $\lambda$-abstraction and to provide a term model that cover multi-rooted rules to express side-effect operations. This will provide a sound mathematical basis for the Id language. It will also be interesting to investigate the suitability of GRSs as an intermediate language for other classes of languages, such as logic languages and imperative languages.

This work was done at the Laboratory for Computer Science at MIT, and at INRIA. Funding for this work has been provided in part by the Advanced Research Projects Agency of the U.S. Department of Defense under the Office of Naval Research contracts N00014-84-K-0099 (MIT) and N0039-88-C-0163 (Harvard), and in part by ESPRIT contract SEMAGRAPH/BRA 3074 (INRIA).

## REFERENCES

[AA89] Z.M. Ariola and Arvind. P-TAC: A Parallel Intermediate Language. In *Proc. ACM Conference on Functional Programming Languages and Computer Architecture, London*, September 1989.

[AA91a] Z.M. Ariola and Arvind. A Syntactic Approach to Program Transformations. In *Proc. ACM SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulation, Yale University, New Haven, CT*, June 1991.

[AA91b] Z.M. Ariola and Arvind. Compilation of Id. In *Proc. of the Fourth Worskshop on Languages and Compilers for Parallel Computing, Santa Clara, California, Springer-Verlag LNCS 589*, August 1991.

[ACCL90] M. Abadi, L. Cardelli, P.-L. Currien, and J.-J. Lévy. Explicit Substitutions. In *Proc. ACM Conference on Principles of Programming Languages, San Francisco*, January 1990.

[ANP89] Arvind, R. S. Nikhil, and K. K. Pingali. I-Structures: Data Structures for Parallel Computing. *ACM Transactions on Programming Languages and Systems*, 11(4), October 1989.

[Ari92] Z.M. Ariola. *An Algebraic Approach to the Compilation and Operational Semantics of Functional Languages with I-structures.* PhD thesis, Harvard University, June 1992.

[BBvE+87] H.P. Barendregt, T.H. Brus, M.C.J.D van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.O. van Leer, M.J. Plasmeijer, and M. Ronan Sleep. Towards an Intermediate Language based on Graph Rewriting. In *Proceedings of the PARLE Conference, Eindhoven, The Netherlands, Springer-Verlag LNCS 259*, June 1987.

[BvEG+87] 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 *Proceedings of the PARLE Conference, Eindhoven, The Netherlands, Springer-Verlag LNCS 259*, pages 141–158, June 1987.

[BvEvLP87] T. Brus, M.C.J.D van Eekelen, M.O. vam Leer, and M.J. Plasmeijer. Clean - A language for Functional Graph Rewriting. In *Proc. ACM Conference on Functional Programming Languages and Computer Architecture, Portland, Oregon, Springer-Verlag LNCS 274*, 1987.

[Cur86] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Research Notes in Theoretical Computer Science. Pitman, London, 1986.

[Cur91] P.-L. Curien. An Abstract Framework for Environment Machines. *Theoretical Computer Science*, 82, 1991.

[Hin77] R. Hindley. Combinator Reductions and Lambda Reduction Compared. In *Zeitschr. f. math. Logik und Grundlagen d. Math. Bd. 23, S. 169-180*, 1977.

[HL89] T. Hardin and J.-J. Lévy. A Confluent Calculus of Substitution. In *France-Japan Artificial Intelligence and Computer Science Symposium, Izu*, 1989.

[Kat90] V.K. Kathail. *Optimal Interpreters for Lambda-calculus Based Funtional Languages.* PhD thesis, Dept. of Electrical Engineering and Computer Science, MIT, May 1990.

[Ken90] R. Kennaway. Implementing Term Rewrite Languages in Dactl. *Theoretical Computer Science*, 1, 1990.

[KKSdV91] J.R Kennaway, J.W. Klop, M.R Sleep, and F.J. de Vries. Transfinite Reductions in Orthogonal Term Rewriting Systems. In *Proc. RTA '91, Springer-Verlag LNCS*, 1991.

[Lé78] J.-J. Lévy. *Réductions Correctes et Optimales dans le Lambda-Calcul.* PhD thesis, Universite Paris VII, October 1978.

[Mar91] L. Maranget. Optimal Derivations in Weak Lambda-calculi and in Orthogonal Term Rewriting Systems. In *Proc. ACM Conference on Principles of Programming Languages, Orlando, Florida*, January 1991.

[Wad71] C. Wadsworth. *Semantics And Pragmatics Of The Lambda-Calculus.* PhD thesis, University of Oxford, September 1971.

[Wel75] P.H Welch. Continuous Semantics and Inside-out Reductions. In *λ-Calculus and Computer Schience Theory, Italy (Springer-Verlag LNCS 37)*, March 1975.