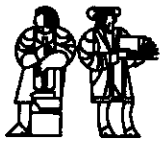
LABORATORY FOR
COMPUTER SCIENCE



MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

Contextual Rewriting System

Computation Structures Group Memo 323

DRAFT

Zena M. Ariola
Arvind

This is a preliminary report on work in progress.

This report describes research done at the Laboratory for Computer Science of the Massachusetts Institute of Technology. Funding for this work has been provided in part by the Advanced Research Projects Agency of the Department of Defense under the Office of Naval Research contract N00014-89-J-1988 (MIT) and N0039-88-C-0163 (Harvard).

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Contextual Rewriting System

Zena M. Ariola
Aiken Computational Laboratory
Harvard University

Arvind
Laboratory for Computer Science
Massachusetts Institute of Technology

December 28, 1990

1 Introduction

Traditionally the operational semantics of functional languages has been given by translating them to the λ-calculus [2] extended with constants [9]. The evaluation of a term is then explained in terms of a reduction process, where at each step a reducible expression (*i.e.* redex) is chosen according to a deterministic reduction strategy and it is reduced following a set of axiom schemes and inference rules.

In the literature functional languages are then divided, according to the evaluation strategy, into two classes: strict and non-strict languages. We strongly beleive that any classification of languages should not be based on the evaluation strategy, because we regard "strictness" as a property and the evaluation strategy as a possible mean to guarantee that property holds. Thus, for example, non-strictness is erroneously taken as synonymous of normal-order reduction, and strictness as synonymous of applicative-order reduction. We beleive that non-strictness could also be achieved using an applicative model of evaluation. A part from the above clarification we recognize that among the

different strategies two of them emerge: applicative and normal order.

Wadsworth proposed in 1971 graph reduction [13], in order to bring together the advantages of both the applicative and the normal order evaluation. The expression to be evaluated is represented as a graph instead of a linear text string, this allows sharing of identical terms by using pointers. More recently a new graph structure which allow sharing of "contexts", has been proposed [5].

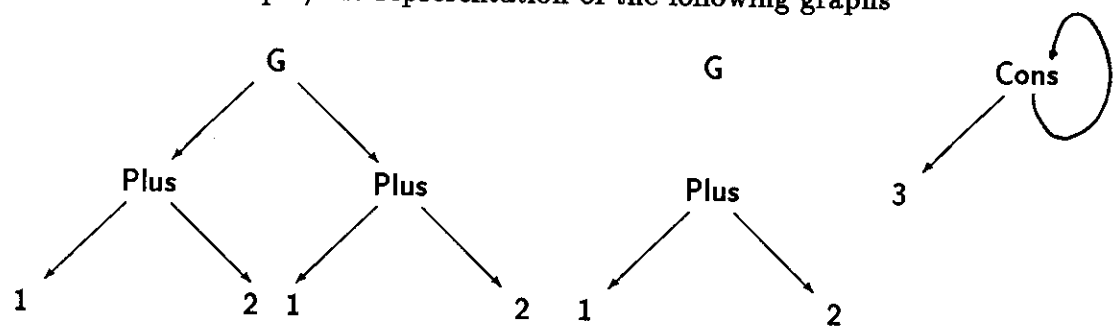
It is clear that if we want to have a computational model which is also close to the implementation, the λ-calculus is not the right choice anymore. Moreover, if we want to consider not only functional languages but also a wider class of languages, such as languages which have a flavor of logic languages, that is, variables can be *consistently* updated, then moving to a new intermediate language other than λ-calculus is not just a matter of choice anymore. Several languages have already been proposed for graph rewriting [3], [6], [12]. The theory supporting graph rewritings has been developed by [10] and by [11]. However, we think that none of them try to investigate *why* the λ-calculus is inadequate. The answer to this question is what this paper is about. We want to propose a new calculus, the $\lambda_{\mathbf{B}}$-calculus, which is the λ-calculus with the block construct as a first class citizen. This entails a new reduction system, which we call **Contextual Reduction System**, which is based on string reduction and it does not preclude any possibility of argument sharing. We beleive that such a system is also suitable to describe languages, like Id [8], which can not be regarded as term rewriting systems.

In Section 2 we introduce the system and following Barendregt [4] we show its soundness and completness with respect to an Orthogonal rewriting system. In Section 3 we introduce the $\lambda_{\mathbf{B}}$-calculus.

2 Contextual Reduction System

2.1 Syntax of Terms

It's seems quite natural that a way of representing a graph textually is by associating an identifier to each node, and writing down all the interconnections as a block, and a way of giving names to subexpressions of an expression is to turn that expression into a block. Therefore, the essential features that a language has to provide in order to be suitable to describe graphs reduction, *i.e.* argument sharing[1], is the block construct and a notion of "value". For example, the representation of the following graphs



is given by the terms below

$$\{\ t = \mathsf{G}(t_1, t_2);\quad t_1 = \mathsf{Plus}(1,2);\quad t_2 = \mathsf{Plus}(1,2);\quad \mathsf{In}\ t\}$$

$$\{\ t = \mathsf{G}(t_1, t_1);\quad t_1 = \mathsf{Plus}(1,2);\quad \mathsf{In}\ t\}$$

$$\{\ x = \mathsf{Cons}(3, x);\quad \mathsf{In}\ x\}$$

Following Barendregt [4] the first two graphs are not equivalent (rooted isomorphism), analogoulsy the corresponding first two terms are not equivalent.

Consider two more examples

[1] Argument sharing is the only sharing we worry about in this paper

the corresponding terms are

$$\{\ t \ = \ \mathsf{A}(t_1);\quad t_1 \ = \ \mathsf{A}(t);\quad \mathsf{In}\ t\}$$

$$\{\ t \ = \ \mathsf{A}(t);\quad \mathsf{In}\ t\}$$

The above two terms are not equivalent, however, their corresponding unravelling are equivalent.

By the previous examples it should be clear that the terms of interest in a contextual reduction system (CRS) are such that every subexpression has a name, this format recall the three-address-code used as intermediate language by compilers [1]. The terms of both a TRS and a CRS are described by the grammars of Figure 1 and of Figure 2 respectively, with *Term* as start symbol.

$$Term ::= Variable \mid \mathsf{F}^0 \mid \mathsf{F}^n\ (Term_1, \cdots, Term_n)$$

Figure 1: Syntax of terms of a TRS

Notice that we do not assume any textual order between bindings (b_i), that is, $b_i;\ b_j \ \equiv \ b_j;\ b_i$.

We define two translation procedures, `TE` and `Fl` respectively. The first one takes a TRS term and produces the corresponding CRS term; the second one takes a CRS term and produces the corresponding TRS term (*i.e.* its unravelling version).

All the variables that appear on the left-hand-side of the translation functions are meta-variables that range over appropriate syntactic categories. By convention, we use capital letters for meta-variables and small letters for terms variables. All variables that

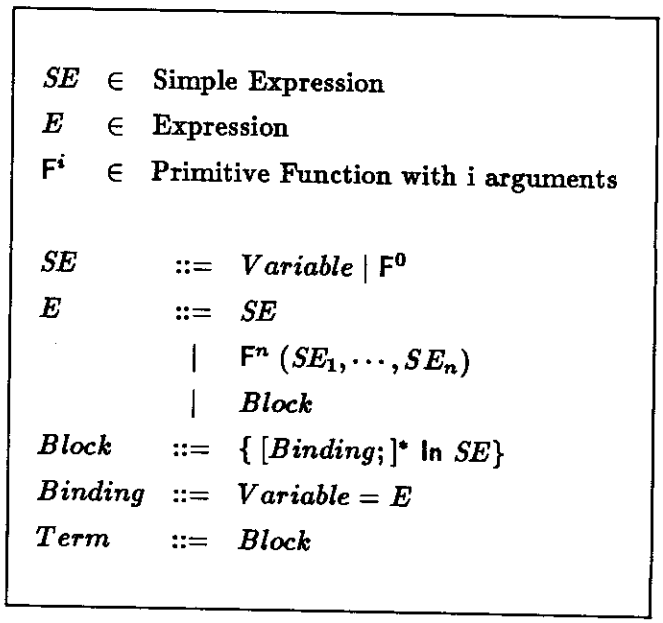


Figure 2: Syntax of terms of a CRS

appear on the right-hand-side of the rules are either meta-variables or "new" term variables. We will make use of the following convention regarding meta-variables:

$$\begin{aligned} X_i, &\in \mathit{Variable} \\ E_i &\in \mathit{Expression} \\ B_i, &\in \mathit{Binding} \end{aligned}$$

Their description is as follows:
On the set of terms we define two notions of syntactical equivalence, α-equivalence and tree-equivalence respectively.

Definition 2.1 (α-equivalence) *Two terms M and N are said to be α-equivalent if each can be transformed into the other by consistent renaming of bound variables.*

Definition 2.2 (tree-equivalence) *Two terms M and N are said to be tree-equivalent if* $\mathtt{Fl}[\![M]\!] \equiv_\alpha \mathtt{Fl}[\![N]\!]$.

$$\mathtt{TE} : \mathit{TRS\ term} \rightarrow \mathit{CRS\ term}$$

$$\mathtt{TE}[\![X]\!] = X$$

$$\mathtt{TE}[\![\mathsf{F}^0]\!] = \mathsf{F}^0$$

$$\mathtt{TE}[\![\mathsf{F}^n\ (E_1, \cdots, E_n)]\!] = \{\ t_1 = \mathtt{TE}[\![E_1]\!];$$
$$\vdots$$
$$t_n = \mathtt{TE}[\![E_n]\!];$$
$$t = \mathsf{F}^n\ (t_1, \cdots, t_n);$$
$$\mathsf{In}\ t\}$$

Figure 3: $\mathtt{TE} : \mathit{TRS\ term} \rightarrow \mathit{CRS\ term}$

$$\mathtt{F1}\ [\![\{B_1; \cdots; B_n\ \ \mathsf{In}\ X\}]\!] = \mathtt{F}\ [\![X]\!]\ (B_1 : \cdots : B_n : \mathsf{nil})$$

$$\mathtt{F} : \mathit{CRS\ term} \rightarrow \mathit{Environment} \rightarrow \mathit{TRS\ term}$$

$$\mathtt{F}\ [\![\{B_1; \cdots; B_n\ \ \mathsf{In}\ X\}]\!]\ \rho = \mathtt{F}\ [\![X]\!]\ (B_1 : \cdots : B_n : \rho)$$

$$\mathtt{F}\ [\![X]\!]\ \rho = \begin{cases} X & \text{if } X \text{ is not found in } \rho \\ \mathtt{F}\ [\![E]\!]\ \rho & \text{otherwise} \quad \text{where E = lookup } X\ \rho \end{cases}$$

$$\mathtt{F}\ [\![\mathsf{F}^n\ (X_1, \cdots, X_n)]\!]\ \rho = \mathsf{F}^n\ ((\mathtt{F}\ [\![X_1]\!]\ \rho), \cdots, (\mathtt{F}\ [\![X_n]\!]\ \rho))$$

$$\mathtt{F}\ [\![\mathsf{F}^0]\!]\ \rho = F^0$$

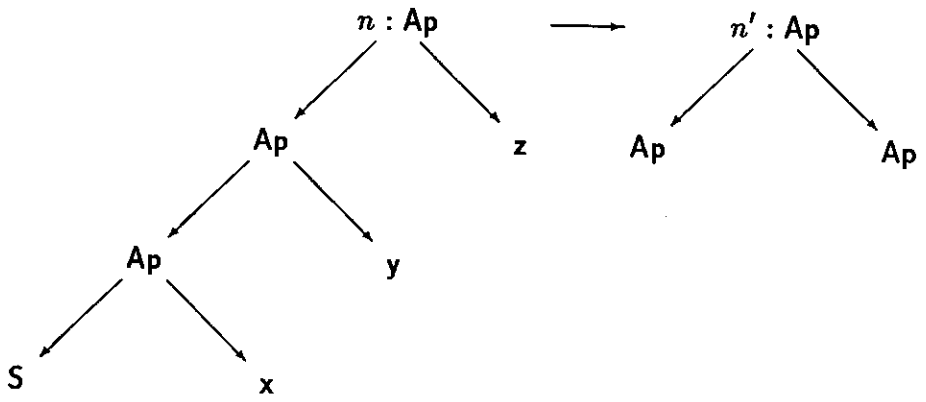
Figure 4: $\mathtt{F1} : \mathit{CRS\ term} \rightarrow \mathit{TRS\ term}$

2.2 Example of reduction

We will first introduce the system through an example. Consider the rule

$$\mathsf{Ap}\,(\mathsf{Ap}\,(\mathsf{Ap}\,(\mathsf{S},\,x),\,y),\,z) \longrightarrow (\mathsf{Ap}\,(\mathsf{Ap}\,(x,\,z),\,\mathsf{Ap}\,(y,\,z))$$

which expressed in Barendregt's graph notation is:



Intuitively, applying this rule consists in allocating three new nodes, which correspond to the three application nodes on the rhs of the rule, and rederecting all the pointers to the node corresponding to n to the node corresponding to n'. This idea is also expressed as rewriting n to n'. Notice that the subgraph $g \equiv \mathsf{Ap}\,(\mathsf{Ap}\,(\mathsf{S},\,x),\,y)$ does not change during the application of this rule. We call g the context or the precondition of the corresponding CRS rule. Since the term structure of a CRS does not allow complex terms we write the graph g as $W_1 = \mathsf{Ap}\,(\mathsf{S},\,X)$; $W_2 = \mathsf{Ap}\,(W_1,\,Y)$. Thus, we write the left-hand-side of the S-rule as

$$\frac{\begin{array}{rcl} W_1 & = & \mathsf{Ap}\,(\mathsf{S},\,X) \;\; | \\ W_2 & = & \mathsf{Ap}\,(W_1,\,Y) \end{array}}{\mathsf{Ap}\,(W_2,\,Z)}$$

The separator between the two bindings indicates that we do not require any specific textual order between them.

In order to represent the right-hand-side of the S-rule we only have to express the notion of allocating new nodes, this simply corresponds to the introduction of new variables name

with their corresponding definition. The S-rule, thus, becomes:

$$S_B\ rule: \quad \frac{\begin{array}{rcl} W_1 & = & \mathsf{Ap}\,(\mathsf{S},\ X)\ \mid \\ W_2 & = & \mathsf{Ap}\,(W_1,\ Y) \end{array}}{\begin{array}{rcl} \mathsf{Ap}\,(W_2,\ Z) \longrightarrow \{\ t_1 & = & \mathsf{Ap}\,(X,\ Z); \\ t_2 & = & \mathsf{Ap}\,(Y,\ Z); \\ t & = & \mathsf{Ap}\,(t_1,\ t_2); \\ \multicolumn{3}{r}{\mathsf{In}\ t\}} \end{array}}$$

Following Barendregt's example we show the contextual reduction of the term M given below:

$$\begin{array}{l} \{\ \boxed{\begin{array}{rcl} w_1 & = & \mathsf{Ap}\,(\mathsf{S},\ \mathsf{P}); \\ w_2 & = & \mathsf{Ap}\,(w_1,\ \mathsf{Q}); \end{array}} \\ \quad w_3 = \underbrace{\mathsf{Ap}\,(w_2,\ \mathsf{R});}_{\rho} \\ \quad w_4 = \mathsf{G}\,(w_1,\ w_3); \\ \quad\ \mathsf{In}\ w_4\} \end{array}$$

M contains two subterms that match the subgraph g, (the two bindings in the box), and a subterm (ρ), that matches the lhs of the S-rule, thus, we call ρ a redex. The matching give rise to the following instantiation of meta-variables:

$$W_1 = w_1,\ W_2 = w_2,\ X = \mathsf{P},\ Y = \mathsf{Q},\ Z = \mathsf{R}$$

Thus, M rewrites, according to the $S_{\mathbf{B}}$ rule, to

$$\begin{array}{l} \{\ w_1 = \mathsf{Ap}\,(\mathsf{S},\ \mathsf{P}); \\ \quad w_2 = \mathsf{Ap}\,(w_1,\ \mathsf{Q}); \\ \quad w_3 = \{\ t_1 = \mathsf{Ap}\,(\mathsf{P},\ \mathsf{R}); \\ \qquad\quad t_2 = \mathsf{Ap}\,(\mathsf{Q},\ \mathsf{R}); \\ \qquad\quad t = \mathsf{Ap}\,(t_1,\ t_2); \\ \qquad\quad \mathsf{In}\ t\} \\ \quad w_4 = \mathsf{G}\,(w_1,\ w_3); \\ \quad\ \mathsf{In}\ w_4\} \end{array}$$

2.3 The Context Sensitive Rewrite Rules of Combinatory Logic

We now present the set of rewrite rules for Combinatory logic. All the variables that appear on the left-hand-side of the rules are meta-variables that range over variables or constants only. The new variables, t_i, that appear on the RHS of a rule represents new variables.

$$\frac{\begin{array}{rcl} W_1 & = & \mathsf{Ap}\,(S,\ X) \quad | \\ W_2 & = & \mathsf{Ap}\,(W_1,\ Y) \end{array}}{\begin{array}{rcl} \mathsf{Ap}\,(W_2,\ Z) & \longrightarrow & \{\ t_1 \ = \ \mathsf{Ap}\,(X,\ Z); \\ & & \quad t_2 \ = \ \mathsf{Ap}\,(Y,\ Z); \\ & & \quad t_3 \ = \ \mathsf{Ap}\,(t_1,\ t_2); \\ & & \quad \mathsf{In}\ t_3\} \end{array}}$$

$$\frac{W \ = \ \mathsf{Ap}\,(K,\ X)}{\mathsf{Ap}\,(W,\ Y) \ \longrightarrow \ X}$$

2.4 Formal Definition of Contextual Rewriting

A Contextual Reduction System is a pair (A(F), R) , where A is a set of terms defined over a signature F, and R is a set of context sensitive rules. Context sensitive rules are a kind of rule whose applicability does not depend upon the structure of the term only. (Notice that left-linearity does not apply to this kind of system, see discussion in [4].

The following is an example of a context sensitive rule[2]:

$$\frac{X = \mathsf{Cons}\,(X_1,\ X_2)}{\mathsf{Head}\,(X) \ \longrightarrow \ X_1}$$

where the binding $X = \mathsf{Cons}\,(X_1,\ X_2)$ over the line denotes a precondition, and X, X_1, and X_2 denote meta-variables. A way of reading the above rule is that $\mathsf{Head}\,(X)$ can be rewritten to X_1 if the binding $\mathsf{Cons}\,(X_1,\ X_2)$ occurs in the context of $\mathsf{Head}\,(X)$.

[2]The corresponding TRS rule is "$\mathsf{Head}\ (\mathsf{Cons}\ X_1\ X_2) \longrightarrow X_1$"

A context is a term with an arbitrary number of $\Box_E$, expression holes, and $\Box_B$, binding holes. Formally contexts are terms generated by the grammar of Figure 2, where $\Box_E$ and $\Box_B$ are added to *SE* and *Bindings*, respectively. Thus, given a context $C[\Box_B, \Box_E]$, and substitution, σ, for the meta-variables X, X_1 and X_2 such that

$$C[(X = \mathsf{Cons}\ (X_1,\ X_2))^\sigma,\ (\mathsf{Head}\ (X))^\sigma]$$

is a term, we say

$$C[(X = \mathsf{Cons}\ (X_1,\ X_2))^\sigma,\ (\mathsf{Head}\ (X))^\sigma] \longrightarrow C[(X = \mathsf{Cons}\ (X_1,\ X_2))^\sigma,\ X_1^\sigma] \qquad (1)$$

where $\longrightarrow$ stands for "rewrites to". $\longrightarrow$ is the reduction relation induced by the above rule. From (1) we can observe that the precondion of the rule is not affected by the rewriting, however, the precondition could affect the outcome of the rewriting.

Definition 2.3 (Context sensitive rule) *A rule is an ordered set of preconditions, $P_1 \cdots P_n$, and a left-hand-side, l, and a right-hand-side, r, and is written as*

$$\frac{P_1 \mid \cdots \mid P_n}{l \longrightarrow r}$$

where P_i is a binding and l and r are expressions.

A redex in a CRS is defined as follows:

Definition 2.4 (Redex) *Given a rule* $\dfrac{P_1 \mid \cdots \mid P_n}{l \longrightarrow r}$ *and a term $M \equiv C[B_1, \cdots, B_n, E]$, E is said to be a* **redex** *iff*

(1) $E \equiv l^\sigma$;

(2) $B_j \equiv P_j^\sigma\ \forall\ j,\ 1 \leq j \leq n$;

where σ is a substitution.

Definition 2.5 (Notions of reductions) *Given terms M,N $\in$ A and a rule $\tau \in R$, then M reduces to N in one step ($M \underset{\tau}{\longrightarrow} N$), iff $M \equiv C[P_1^\sigma,\ \cdots, P_n^\sigma,\ l^\sigma]$, and $N \equiv C[P_1^\sigma,\ \cdots, P_n^\sigma,\ r^\sigma]$.*

The one-step rewriting relation is $\longrightarrow_R = \bigcup_{\tau \in R}(\underset{\tau}{\longrightarrow})$.
The transitive reflexive closure of $\longrightarrow_R$ is written as $\twoheadrightarrow_R$.

2.5 Orthogonal CRS

Analogously to TRS's we want to define a class of CRS's, the Orthogonal Contextual Rewriting System, that have nice properties, as confluence.

We need to extend the notion of overlapping patterns [7] with a notion of *non-interference*, which is defined as follows

Definition 2.6 (Interence) *i-rule* **interferes** *with j-rule iff:*

1. *the application of i-rule can invalidate the precondition of j-rule; or*
2. *the pattern of i-rule overlaps with the pattern of j-rule.*

As an example consider the TRS:

$$\mathsf{L}\,(\mathsf{L}\,(x)) \longrightarrow 0$$

which is not Orthogonal due to overlapping of patterns.
The corresponding CRS rule is

$$\frac{Y = \mathsf{L}(X)}{\mathsf{L}(Y) \longrightarrow 0}$$

Consider the term

$$\begin{array}{l}\{\ y_1 = \mathsf{L}(x);\\ \quad y_2 = \underbrace{\mathsf{L}(y_1);}_{\rho_1}\\ \quad y = \underbrace{\mathsf{L}(y_2);}_{\rho_2}\\ \quad \mathsf{In}\ y\}\end{array}$$

Notice that the ρ_1-reduction destroys the precondition of redex ρ_2; we will say that the two reductions interfere.

Definition 2.7 (Disjoint) *Two redexes r_1 and r_2 are said to be disjoint iff there is no data dependency between them. That is*

$$FV(r_i) \cap BV(r_j) = \emptyset \quad i \neq j \ \wedge \ i,j = 1,2$$

where $FV(e)$ represents the free variables of expression e, and $BV(e)$ represents the variable that bounds e.

Notice that non-interfering redexes may not be disjoint, as in the CRS shown below:

$$F(X) \longrightarrow 0 \quad I(X) \longrightarrow X$$

Consider the term

$$\begin{aligned} \{\ y &= \underbrace{I(x);}_{r_2} \\ y_1 &= \underbrace{F(y);}_{r_1} \\ &\text{In } y_1\} \end{aligned}$$

r_1 and r_2 are non-interfering, however they are not disjoint, because there is a data dependency between them. In the presence of non-disjoint redexes, a reduction may remove a data dependency, that translates to deleting a redex. As in the example above, r_1 - reduction deletes redex r_2.

Definition 2.8 (Orthogonality) *A CRS (C,R) is Orthogonal if all rules in R are non-interfering.*

For example, the CRS associated to Combinatory Logic is Orthogonal.

Theorem 2.9 *An Orthogonal CRS, (C,R), is subcommutative (upto dead code elimination).*

Proof: The proof is similar to the proof of WCR for the underlying version of an Orthogonal Rewriting System [7].
Suppose there are two redexes r_1 and r_2, respectively. By definition of Orthogonal they can appear as follows:

(i) **r_1 and r_2 are disjoint** Trivial.

(ii) **r_1 depends on r_2** (ii.a) **r_2 does not depends on r_1** Trivial.

(ii.b) **r_2 does depends on r_1** Suppose they both destroy each other. In this case, we have a cyclic dependency and since they both destroy each other it means that there are no other references to either r_1 or r_2, therefore they are garbage, and the dead-code elimination phase will eliminate them.

■

The important point to underline is that we never have to consider the "duplication" case, that is, redexes in an Orthogonal CRS are never *duplicated*, or *modified*, they could only by *destroyed.*

Corollary 2.10 *An Orthogonal CRS is CR.*

Proof: By induction on the number of reduction steps. ■

We now describe the correspondence between a TRS and the corresponding CRS (named CTRS).

Definition 2.11 *Given a TRS (T,R) its corresponding CRS, is (Fl(T),R_c). Where Fl(T) = { t | Fl(t) ∈ T } and R_c are the corresponding context sensitive rules.*

Lemma 2.12 *Given a non-overlapping rule the corresponding CTRS rule is non-interfering.*

Lemma 2.13 *Given an Orthogonal TRS the corresponding CTRS is Orthogonal.*

2.6 Soundness and Completness

With respect to Orthogonal TRS's context reduction is sound and complete.

Definition 2.14 (Soundness) *A CTRS (Fl(T),R_c) is reducible to a TRS (T,R) (soundness) if*

$$\forall\, t \in Fl(T),\ t \text{ has NF } t' \implies Fl(t) \text{ has normal form } Fl(t')$$

and if t does not have a NF then Fl(t) does not either.

Definition 2.15 (Completness) *A TRS (T,R) is reducible to a CTRS (Fl(T),R_c) (completness) if*

$$\forall\, t \in T,\ t \text{ has NF } t' \implies TE(t) \text{ has normal form } TE(t')$$

and if t does not have a NF then TE(t) does not either.

Notice that we only consider CTRS, this means that all terms are acyclic.

Lemma 2.16 *Given a CTRS* $(A(\mathsf{F}), R_c)$, $\forall\ t \in A(\mathsf{F})$

$$t \underset{r_1}{\longrightarrow} t' \implies \mathtt{Fl}(t) \longrightarrow\!\!\!\!\!\rightarrow \mathtt{Fl}(t')$$

Proof: If $t \underset{r_1}{\longrightarrow} t'$ it means that Fl(t) will contain at least one copy of r_1 (the number is finite). Since r_1-reductions do not overlapp it is obviuos that flattening commutes with r_1-reductions. ■

Theorem 2.17 *Every Orthogonal CTRS is reducible to a TRS.*

Proof: By induction. ■

Notice that is not true that every CRS is sound, as shown by the next counterexample, which is taken from [4].

$$A\,(x) \longrightarrow B\,(x)$$

Consider the term $\{y = A(y);\ \mathsf{In}\ y\}$ which has the normal form $\{y = B(y);\ \mathsf{In}\ y\}$, while the corresponding unravelling does not have one.

Notice however that we intend to overcome this problem, by consider a different notion of equivalence between term, as for example, a notion of observational equivalence based on printed values.

Theorem 2.18 *Every Orthogonal TRS is reducible to a CTRS.*

Proof: By considering what it means to be Orthogonal. ■

Notice that completness does not hold for general TRS's. For a counterexample, consider the following TRS, which is similar to the example in [4]

$D\,(x) \longrightarrow A(x,\ x)$
$A\,(\,B(x,\ y),\ C(x,\ y)) \longrightarrow 0$
$B\,(x,\ y) \longrightarrow C\,(x,\ y)$

$$C\,(x,\ y) \quad \longrightarrow \quad B\,(x,\ y)$$

Consider the term $D\ (B\ (x,\ y))$, it has a normal form, while in the corresponding CTRS the corresponding term does not have a normal form. To understand the problem consider the terms in the CRS and TRS, respectively,

$$\begin{array}{rcll} \{\ z & = & \mathsf{B}(x,y); & \mathsf{A}(\mathsf{B}(x,y),\mathsf{B}(x,y)) \\ z_1 & = & \mathsf{A}(z,z); & \\ \multicolumn{4}{r}{\mathsf{In}\ z_1\}} \end{array}$$

Notice that the first term contains one redex, while the second one contains two distinct redexes, which can be reduced separately. That is, the term can reduce to $\mathsf{A}(\mathsf{B}(x,y),\mathsf{C}(x,y))$ If there exists a rule that can tell this transiction then clearly it will not be applicable in the corresponding CRS. Notice however that in this case the TRS can't be Orthogonal.

Notice some curious facts.

Fact 2.19 *TRS* $\not\models$ *SN* $\not\Longleftrightarrow$ *CTRS* $\not\models$ *SN.*

For a counterexample, consider the following TRS taken from [7]

$$\begin{array}{l} or\ (x,\ y) \ \rightarrow \ x \\ or\ (x,\ y) \ \rightarrow \ y \\ F\ (0,\ 1,\ x) \ \rightarrow \ F(x,\ x,\ x) \end{array}$$

It's easy to verify that the above is not SN while its corresponding CTRS is indeed SN.

3 $\lambda_{\mathbf{B}}$-calculus

The $\lambda_{\mathbf{B}}$-calculus is the λ-calculus plus the block construct. The rational being that the new calculus should be a more appropriate intermediate language for functional languages than the λ-calculus. Moreover, it is suitable also for languages, like Id [8], that are not referential transparent, but are still functional. The λ-calculus syntax is described in

$$
\begin{array}{lll}
F & ::= & \mathit{Integers} \mid \mathit{Booleans} \mid \mathsf{Nil} \mid \mathsf{Cons} \mid \mathsf{Head} \mid \mathsf{Tail} \mid \mathsf{Y} \\
\mathit{Term} & ::= & \mathit{Variable} \\
 & \mid & F \\
 & \mid & \mathit{Term}\ \mathit{Term} \\
 & \mid & \lambda\, x\, .\, \mathit{Term}
\end{array}
$$

Figure 5: Syntax of λ-calculus

$$
\begin{array}{lll}
F^n & ::= & \mathit{Integer} \mid \mathit{Boolean} \mid \mathsf{Nil} \mid \mathsf{Cons} \mid \mathsf{Head} \mid \mathsf{Tail} \mid \mathsf{Y} \\
SE & ::= & \mathit{Variable} \mid F^0 \\
E & ::= & SE \\
 & \mid & \lambda\, \mathit{Variable}\, .\, E \\
 & \mid & F^n\ (SE_1, \cdots, SE_n) \\
 & \mid & \mathit{Block} \\
\mathit{Block} & ::= & \{\ [\mathit{Binding};]^*\ \mathsf{In}\ SE\} \\
\mathit{Binding} & ::= & \mathit{Variable} = E \\
\mathit{Term} & ::= & \mathit{Block}
\end{array}
$$

Figure 6: Syntax of terms of $\lambda_{\mathbf{B}}$-calculus

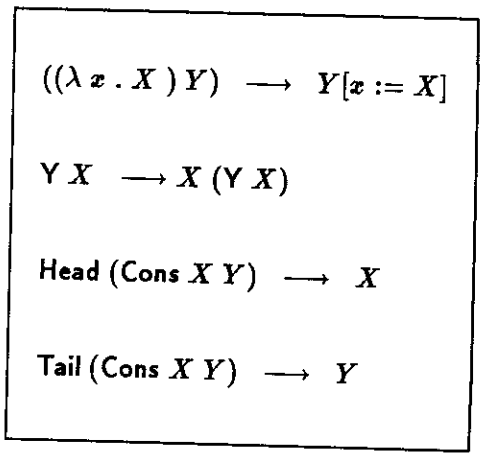


Figure 7: Rewrite rule for the λ-calculus

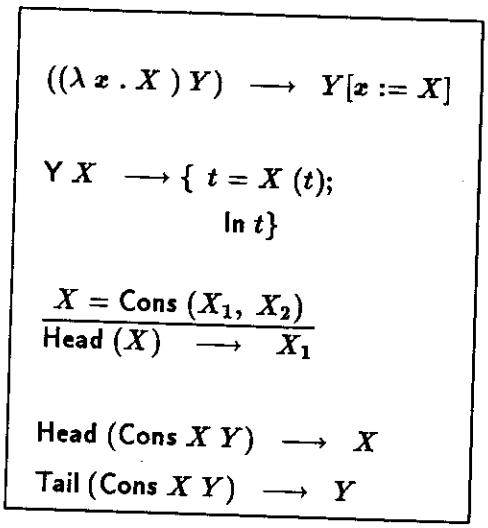


Figure 8: Rewrite rule for the $\lambda_{\mathbf{B}}$-calculus

Figure 5, while the $\lambda_{\mathbf{B}}$-calculus syntax is given in Figure 6. The rewrite rules of both the λ-calculus and of the $\lambda_{\mathbf{B}}$-calculus are given in Figure 7 and Figure 8 respectively.

References

[1] A. A.V., J. Ullman, and R. Sethi. *Compilers: Principles, Techniques, Tools.* Lodon, Addison-Wesley, 1986.

[2] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, Amsterdam, 1984.

[3] H. P. Barendregt, B. T.H., van Eekelen M.C.J.D, G. J.R.W., K. J.R., van Leer M.O., P. M.J., and S. M. Ronan. Towards an intermediate language based on graph rewriting. In *Proceedings of the PARLE Conference, Eindhoven, The Netherlands, Springer-Verlag LNCS 259*, June 1987.

[4] 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.

[5] V. K. Kathail. Optimal interpreters for lambda-calculus based funtional languages, May 1990. Ph.D. thesis, Dept. of Electrical Engineering and Computer Science, MIT.

[6] R. Kennaway. Implementing term rewrite languages in dactl. *Theoretical Computer Science*, 1, 1990.

[7] J. Klop. Term rewriting systems. Course Notes, Summer course organized by Corrado Boehm, Ustica,Italy, September 1985.

[8] R. S. Nikhil. Id (version 88.1) reference manual. Technical Report CSG Memo 284, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, August 1988.

[9] S. L. Peyton Jones. *The implementation of Functional Programming Languages.* Prentice-Hall International, Englewood Cliffs, N.J., 1987.

[10] J. C. Raoul. On graph rewritings. *Theoretical Computer Science*, 1, 1984.

[11] J. Staples. Computation on graph-like expressions. *Theoretical Computer Science*, 1, 1980.

[12] B. T., M. van Eekelen, M. vam Leer, and M. 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.

[13] C. Wadsworth. *Semantics And Pragmatics Of The Lambda-Calculus.* Ph.D. thesis, University of Oxford, Semtember 1971.

Contents