Strictness Analysis by Abstract Reduction in Orthogonal Term Rewriting Systems

Eric Nöcker
University of Nijmegen *

July 1993

Abstract

The strictness analyser in the Concurrent Clean system is based on a technique called abstract reduction. In this paper, we give a theoretical foundation of abstract reduction. The abstract reduction mechanism has a very operational character. Therefore, we will use orthogonal constructor term rewriting systems as the basic model. We prove safety of abstract reduction, and of the approximation techniques that can be applied. Finally, we also indicate how the presented method can be extended to functional languages.

1. Introduction

Strictness analysis is an important technique in the compilation of lazy functional languages. Most strictness analysis techniques are based on abstract interpretation frameworks (e.g. Mycroft [1981], Burn *et al.* [1985], Wadler & Hughes [1987]). However, the strictness analyser in the Concurrent Clean System is based on a technique called *abstract reduction* (Nöcker [1993]). Concurrent Clean (Nöcker *et al.* [1991]) is a lazy functional language based on term graph rewriting (Barendregt *et al.* [1987]). It has all the basic ingredients of a lazy functional language, so the strictness analysis method can also be used for other lazy functional languages. Informally, this technique can be considered as a kind of *abstract* term graph rewriting, i.e. it is a purely operational approach. In contrast to most methods based on abstract interpretation, an infinite domain of values is used. Recursion is not handled with fixed point techniques, but with reduction path analysis. In Nöcker [1993] an informal description is given. In the latter paper it is also shown that the results obtained are very good. This holds both for simple kinds of strictness analysis and for cases where higher order functions or complex data structures are involved. A short introduction to abstract reduction is given in the next section.

Until now there was no satisfying theoretical foundation of this technique. A first start was given by Eekelen *et al.* [1991]. In this paper a theory was developed based on a basic form of abstract interpretation, which used a complete but still infinite variant of the original domain (Goubault & Hankin [1991]). The results, however, were rather limited. The main problems are caused by the complexity of the domain, by the different forms of possible sharing, and by the functional reduction strategy.

In this paper, we will develop a framework that avoids these problems. We will describe abstract reduction for orthogonal term rewriting systems. To some extent, orthogonal term rewriting systems can be used for modeling the behaviour of functional languages. Clearly, we can have no problems caused by sharing. Because of confluence of reduction in orthogonal systems, we don't need a difficult reduction strategy. To be able to discuss a usable form of strictness, we will define a special notion of head normal forms and of neededness. We will prove the basic properties of abstract reduction and of some approximation techniques.

*Faculty of Mathematics and Computer Science, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands E-mail: clean@cs.kun.nl

1.1. Abstract reduction and reduction path analysis

First, we will give an informal and short overview of abstract reduction. A detailed description can be found in Nöcker [1993].

Let Σ be the set of symbols defined by a functional program. The set $\Sigma^{\#}$ of abstract symbols is defined as:

$$\Sigma^{\#} = \{F^{\#} \mid F \in \Sigma\} \cup \{\mathsf{Top}, \mathsf{Bot}, \mathsf{Union}\}$$

In the sequel, we will omit the #'s. The objects in the abstract reduction framework are *graphs* built from the set of abstract symbols. Such graphs can be considered as representations of sets of concrete graphs. As usual, Top represents all expressions, whereas Bot represents non-termination. The Union is used to combine sets. By using Unions and cycles, sets can become very complex. In particular, it is possible to define some special abstract values with which strictness analysis for lists can be done. For example, inf: Cons Top inf represents all infinite lists, and topmem: Union (Cons Top topmem) Nil represents all lists (with the colon symbol sharing of (sub)graphs is expressed).

The reduction process simply mimics concrete reduction as precise as possible. A possible definition is that an abstract reduction step should contain all concrete reduction steps. Let $\rightarrow_\alpha$ be reduction in the abstract domain, and γ be a concretisation function that relates abstract graphs and the sets of concrete graphs they represent. For such a reduction step it should hold that:

$$t_1 \rightarrow_\alpha t_2 \Rightarrow \{g \mid h \rightarrow g, h \in \gamma(t_1)\} \subseteq \gamma(t_2)$$

This requirement, however, is somewhat cumbersome. It is such precise that abstract reductions would very quickly lead to huge abstract terms (to include all the information) or to bad approximations (loosing too much information). In this paper we will formulate a better requirement, based on a notion of needed reductions.

Abstract reduction can be implemented by using the original rewrite rules, that is, no conversion or abstraction of these rules is necessary. A typical property of an abstract reduction sequence is that it is not necessarily terminating. (for example, if it precisely models a concrete reduction sequence). However, by analysing abstract reduction sequences (*reduction path analysis*) it is often possible to optimise or approximate the reductions. Currently, we consider the following three possibilities:

- Top introduction can be used to enforce termination of possibly infinite abstract reduction sequences. It states that at any point during the analysis any term can be replaced by Top. This is the 'worst case' approximation.
- Bottom introduction is used for handling recursive behaviour in reduction sequences. It states that if a value t reduces to a value in which t itself is contained on a needed position, t can equally well be replaced by Bot.
- Cycle introduction is also used for handling recursive behaviour in reduction sequences. It states that if a value t reduces to a value in which t itself is contained, t can equally well be replaced by a cyclic value. Thus, the difference with Bottom introduction is that the neededness condition is not fulfilled.

By way of illustration, we will give some simple examples of abstract reduction steps. Consider the following functions:

```
F (x)   → F (x)
G (x)   → A (G (x))
```

Then, we might have the following abstract reductions (suppose A is a constructor):

```
F (Top)   →α F (Top) →α Bot          (ordinary reduction, then Bot introduction)
F (Top)   →α Top                     (Top introduction)
G (Top)   →α A (G (Top)) →α x: A (x) (ordinary reduction, then cycle introduction)
```

Though it might seem at first sight that an implementation of a strictness analyser based on abstract reduction cannot be efficient, the implementation in the Concurrent Clean system as described in Nöcker [1993] has shown that this is not the case. On the contrary, this analyser is surprisingly fast and finds much strictness information.

1.2. Overview

The remainder of this paper is organised as follows. The next section shortly introduces orthogonal term rewriting systems and some important properties related to such systems. In section 3 and 4 we define the notions of constructor head normal form and $\perp$-reduction. They will be used for defining what strictness means. Also, these notions are related to a specific form of neededness. We define a needed reduction relation which will be the basis for the abstract reduction relation. Section 5 treats *approximations*, with which we will describe the basic form of abstract reduction itself. We prove correctness of abstract reduction and of Bottom introduction. Section 6 gives a more operational interpretation of approximations. It can be used as the start of (a description of) an implementation. Correctness of cycle introduction is proved in section 7. It uses notions from the theory of transfinite reductions. Finally, section 8 concludes. There, we also discuss possible extensions, in particular for functional languages.

2. Basic Notions

In this section we briefly recall the basic notions of orthogonal term rewriting systems, as presented in Klop [1992]. We start by defining some basic notions and notations related to *reduction systems* (normally called *abstract* reduction systems; however, to prevent confusion with the abstract reduction mechanism we want to describe we will simply speak of reduction systems).

2.1. DEFINITION (Reduction Systems). (i) A *reduction system* is a pair $(A, (\rightarrow_\alpha)_{\alpha \in I})$, containing a set of objects A, and a sequence of binary relations $\rightarrow_\alpha$ on A, also called reduction or rewrite relations. The subscript α is submitted if it is clear from the context. The transitive reflexive closure of $\rightarrow$ will be written as $\twoheadrightarrow$, the transitive closure as $\rightarrow^+$. A sequence of exactly k reduction steps $a_0 \rightarrow a_1 \rightarrow \ldots \rightarrow a_k$ where $k > 0$, is denoted as $\rightarrow^k$. A sequence of exactly zero or one reduction step (the reflexive closure) is denoted as $\rightarrow^\varepsilon$. The union of two reduction relations is $\rightarrow_\alpha \cup \rightarrow_\beta$, or more shortly $\rightarrow_{\alpha\beta}$. We will write $a_0 \rightarrow \cdot \rightarrow a_2$ instead of $a_0 \rightarrow a_1 \rightarrow a_2$ if the identity of the intermediate term is not essential.

(ii) A reduction relation $\rightarrow$ is *confluent* or *Church-Rosser* if $\forall a, b, c \in A \; \exists d \in A \; [(a \twoheadrightarrow b \wedge a \twoheadrightarrow c) \Rightarrow (b \twoheadrightarrow d \wedge c \twoheadrightarrow d)]$.

(iii) An object t is in *normal form* if there is no object s such that $t \rightarrow s$.

2.2. DEFINITION (Term Rewriting Systems). A *Term Rewriting System* is a pair (Σ, R) of an *alphabet* or *signature* Σ and a set of *rewrite rules* R.

The alphabet Σ consists of a countable infinite set of variables $(x, y, z \ldots)$ and a non-empty set of *function symbols* $F, G \ldots$ each equipped with a fixed finite arity n (≥ 0), being the number of arguments it can have.

The set of *terms* over Σ is $T(\Sigma)$ and is defined inductively:

(i) $x, y, z, \ldots \in T(\Sigma)$,

(ii) if F is an n-ary symbol and $t_1 \ldots t_n \in T(\Sigma)$, then $F(t_1, \ldots, t_n) \in T(\Sigma)$

For a non-variable term $t = F(t_1, \ldots, t_n)$, F is called the *root symbol*, also written as $root(t)$. We write $t = s$ if t and s are syntactically equal. If a term t contains a variable x, the term is called *open*, and the variable x free. Otherwise, a term is *closed*.

Contexts are terms in $T(\Sigma \cup \{\Box\})$ where the special symbol $\Box$ denotes an *empty place* or *hole*. A context containing n holes is written as $C[\,, \ldots, \,]$. For $t_1 \ldots t_n \in T(\Sigma)$, $C[t_1, \ldots, t_n]$ denotes the result of replacing the holes from left to right by $t_1 \ldots t_n$. Especially, $C[]$ denotes a context with exactly one hole.

A term s is a *subterm* of a term t if $t = C[s]$. We write $s \subseteq t$, and $s \subset t$ if $s \neq t$ (in this case s is a *proper* subterm). Two subterms of one term are disjoint if they are no subterms of each other (with respect to the surrounding term). If a term s appears k ($k \geq 0$) times in a term t, we will also write $t = C[s^k]$. Subterms can be identified more precisely by *occurrences*. An occurrence is a sequence of positive integers, denoted by $u_1.u_2 \ldots u_n$, and where $\bullet$ is the empty sequence. For a term t, the subterm t/u is defined as follows:

$$
\begin{array}{lcll}
t/\bullet & = & t & \\
F(t_1,\ldots,t_n)/u_1.u & = & t_{u_1}/u & \text{if } 1 \le u_1 \le n \\
 & = & \text{undefined} & \text{otherwise}
\end{array}
$$

The set of valid occurrences for a term t is denoted by $O(t)$. The *depth* of a subterm t/u in t is the length of the sequence u. The depth of a term is the maximal depth of all its subterms.

A *rewrite rule* is a pair (l, r) of terms with $l, r \in T(\Sigma)$, such that the left-hand side l is not a single variable and the variables of r are contained in l. A rewrite rule will be written as $\mathbf{f}$: $l \to r$, where $\mathbf{f}$ is the name of the rewrite rule. A *function* F is the set of rewrite rules for which $root(l) = F$. We assume functions to be finite. A *substitution* σ is a map from $T(\Sigma)$ to $T(\Sigma)$ satisfying $\sigma(F(t_1,\ldots,t_n)) = F(\sigma(t_1),\ldots,\sigma(t_n))$. Instead of $\sigma(t)$ we also write t^σ. A *redex* is a tuple ($\mathbf{f}$: $l \to r, \sigma, t$) consisting of a rewrite rule $\mathbf{f}$, a substitution σ, a term t, such that $t = \sigma(l)$. The set of rewrite rules R defines a reduction relation $\to_r$ on $T(\Sigma)$ as follows: $t \to_r s$ if there exists a redex ($\mathbf{f}$: $l \to r, \sigma, v$) and a context $C[]$ such that $t = C[l^\sigma]$ and $s = C[r^\sigma]$. Also the term l^σ is called a redex.

2.3. DEFINITION. (i) A term t is *linear* if no variable occurs more than once. A rewrite rule $\mathbf{f}$: $l \to r$ is *left-linear* if l is a linear term.

(ii) Two terms t and $s \in T(\Sigma)$ are *unifiable* or *overlapping* if there exist substitutions σ_1 and σ_2 such that $\sigma_1(t) = \sigma_2(s)$.

A term rewriting system R is *orthogonal* if:

(i) all rewrite rules are left-linear

(ii) for any two rewrite rules $\mathbf{f}_1$: $l_1 \to r_1$ and $\mathbf{f}_2$: $l_2 \to r_2$

(a) if $\mathbf{f}_1$ and $\mathbf{f}_2$ are different, then l_1 and l_2 are non-overlapping

(b) for all $s \subset l_2$ such that s is not a single variable, l_1 and s are non-overlapping

2.4. THEOREM. *An orthogonal TRS is confluent.*

PROOF. See Klop [1992]. □

Besides confluence, orthogonal term rewrite systems have some other useful properties; we refer to Klop [1992]. From now on we will only consider orthogonal term rewrite systems.

There are other possibilities to express the notion of orthogonality. In this paper we will often use a derived concept, based on the fact that in an orthogonal system a redex cannot contribute to another redex (i.e. there are no critical pairs). Also, we define in a semi-formal way what the residuals of a redex by a reduction are.

2.5. DEFINITION. Let $\Delta = (\mathbf{f}$: $l \to r, \sigma, t)$ be a redex. Let s be a proper subterm of t such that $s = t/u$. This subterm s *contributes* to the redex if l/u is a subterm of l, not being a variable.

2.6. LEMMA. (i) *Let $\Delta = (\mathbf{f}$: $l \to r, \sigma, t)$ be a redex, and $t = C[s]$ such that s doesn't contribute to Δ. Then there is a context $C'[]$ and a $k\,(\ge 0)$ such that for any term $a \in T$: ($\mathbf{f}$: $l \to r, \sigma, C[a]$) is a redex, and $C[a] \to_r C'[a^k]$.*

(ii) *A redex, or a term that can reduce to a redex, doesn't contribute to any redex.*

PROOF. (i) Trivial. (ii) Follows from the orthogonality of the system. □

2.7. DEFINITION. Let t be a term containing a redex Δ. Let t' be the result of the contraction of Δ in t (such that $C[\Delta] \to_r C[r] = t'$). Let $s = t/u$ also be a subterm of t. The set of *descendants* of s after reduction of Δ (notation s/Δ), is defined as follows:

(i) if s and Δ are the same, then $s/\Delta = \emptyset$.

(ii) if s and Δ are disjoint. Then s has exactly one descendant, being equal to s, and $s/\Delta = \{t/u\}$.

(iii) if $\Delta \subset s$. Let $s = C[\Delta]$. Then, s has exactly one descendant, $C[r]$, and $s/\Delta = \{t/u\}$.

(iv) if $s \subset \Delta$. Let $\Delta = C[s]$. If s contributes to the redex Δ, then $s/\Delta = \emptyset$. Otherwise, there is by lemma 2.6 a context $C'[]$ such that, for any term a: $C[a] \rightarrow_r C'[a^k]$ $(k \geq 0)$. Let $\Delta \rightarrow_r C'[s^k]$. Then the k terms s are the descendants of s.

If a subterm is a redex, then also its descendants are redexes (of the same rule). In that case we speak of *residuals*, instead of descendants. The definition may be extended to reduction sequences. The notion of descendant is valid in any left-linear system. The notion of residual is typical for orthogonal term rewriting systems, since in other systems the descendant of a redex is not necessarily a redex.

3. Strictness and $\perp$-reduction

Clearly, a term t is in normal form if it contains no redexes. In term rewriting theory, properties related to normal forms are very important. However, in the context of functional languages and of strictness analysis, the notion of *head* normal forms is more useful.

3.1. DEFINITION. A term $t \in T(\Sigma)$ is in *head normal form (hnf)* if there exists no redex $s \in T(\Sigma)$ such that $t \twoheadrightarrow_r s$.

Why are we interested in head normal forms? Most reduction strategies applied in functional languages are based on reduction to head normal form. Normal form reduction is obtained by repeated application of head normal form reduction. Also in the case of pattern matching, a value will be reduced to head normal form. As a consequence, head normal form strictness is much more important than normal form strictness in implementations of functional languages.

However, head normal form properties can be rather cumbersome in term rewriting systems. Strictness is an example of this. The *intention* of strictness is as follows: a function F is strict in its i-th argument if for any application of F the i-th argument (or a descendant) can (or has to) be reduced to head normal form in order to obtain a head normal form for that application. This simple intention gives already problems. Consider for example the single left-hand side of a function F:

$$\mathsf{F}\ (1) \quad \rightarrow \quad \ldots$$

Maybe one might want F to be strict in its argument. However, for a term $\mathsf{F}\ (\Delta)$ it is still not clear (and generally undecidable) whether or not the redex Δ should be reduced or not: the term is already in head normal form if Δ cannot reduce to 1. Clearly, such a form of strictness is not useful.

This problem is caused by the fact that the head normal form property is in general undecidable. This might be solved by 'completing' the rewrite system, that is, by requiring that every term with as root a function symbol will eventually become a redex (either by restricting the kinds of rewrite systems, or by restricting the set of valid terms by some form of typing). Then, the head normal form property becomes decidable. In most functional languages such a strategy is applied. We will solve this by introducing a special variant of head normal forms.

3.2. DEFINITION. Within an alphabet Σ we will consider two kinds of symbols: a symbol is either a *constructor* or a *function symbol*. The set Σ_F of function symbols contains exactly those symbols being the root of a left-hand side of a rewrite rule. All other symbols are in Σ_C, the set of constructor symbols.

A term rewriting system (Σ, R) is a *function-constructor system* if for all proper subterms s, not being a single variable, of all left-hand sides of R: $root(s) \in \Sigma_C$.

A term t is in *constructor head normal form (chnf)* if $root(t) \in \Sigma_C$. A term t *has* a constructor head normal form if $t \twoheadrightarrow_r t_c$ and t_c is in constructor head normal form. Clearly, constructor head normal form implies head normal form.

We restrict our intention of strictness to constructor head normal forms. The notion of constructor head normal form is more usable. Firstly, it is decidable whether a term is in constructor head normal form. Secondly, this property is more close to what we expect. For example, the above function F becomes strict in its argument. However, also this notion of head normal forms introduces a problem. In the following we want to relate a reduction relation $\rightarrow_\perp$ to strictness-like properties. Let $\perp$ be a new and

special symbol. Then we want that $C[t] \to_\perp C[\perp]$ if t has no constructor head normal form with respect to $\to_r$. Now consider the following function:

$$\mathsf{F}\ (\mathsf{F}\ (1)) \quad \to \quad 3$$

Now we have the following reductions: $\mathsf{F}\ (\mathsf{F}\ (1)) \to_\perp \mathsf{F}\ (\perp) \to_\perp \perp$ and $\mathsf{F}\ (\mathsf{F}\ (1)) \to_r 3$. This means that the combination of $\to_r$ and $\to_\perp$ is not confluent, which is rather problematic.

This problem is caused by the fact that a term without chnf (in the above case $\mathsf{F}\ (1)$) can contribute to a redex. Clearly, this is not the case for function constructor systems. There are more ways to define head normal form like properties (and related rewrite systems). In general, we can probably state that a specific form of strictness is meaningful if it is combined with some decidable form of (head) normal forms in rewrite systems where each term without such a (head) normal form cannot contribute to a redex.

In the following, we are only interested in constructor head normal forms, and thus we will only consider orthogonal function-constructor systems. If considering lazy functional languages, this restriction is not too serious. In section 8 we will indicate how the method can be extended to the world of lazy functional languages. In the remainder of this section, we will formalize the notion of $\perp$-reduction, and prove some useful properties. For a term rewriting system (Σ, R) we will extend the alphabet Σ in various ways. The reduction relations will be related to the original set of rewrite rules R, that is, the set of rewrite rules R remains fixed.

3.3. DEFINITION ($\perp$-terms and $\perp$-reduction). (i) Consider the extra symbol $\perp$. The set $T(\Sigma \cup \{\perp\})$, also denoted as $T_\perp$, is called the set of $\perp$-terms. The reduction relation $\to_r$ is extended to $\perp$-terms in the obvious way.

(ii) The partial ordering $\leq$ on $T_\perp$ is defined inductively as follows:

$$\begin{array}{lcll} \perp & \leq & t & \text{for all } t \in T_\perp. \\ F(t_1, \ldots, t_n) & \leq & F(s_1, \ldots, s_n) & (n \geq 0) \text{ if } t_i \leq s_i \text{ for all } i = 1 \ldots n. \end{array}$$

We write $t < s$ if $t \leq s$ and $t \neq s$.

(iii) $\perp$-reduction, denoted by $\to_\perp$, is defined on $T_\perp$ as $C[t] \to_\perp C[\perp]$ if t has no constructor head normal form with respect to $\to_r$.

In general, it is undecidable whether a term has a chnf. However, in some cases it is safe to perform $\perp$-reductions. For example, if a term t has no redex, and has as root symbol a function symbol, then $t \to_\perp \perp$. Examples of $\perp$-reductions are:

Rewrite rules:

$$\begin{array}{lcl} \mathsf{F}\ (2, \mathsf{x}) & \to & \mathsf{G}\ (\mathsf{x}) \\ \mathsf{G}\ (1) & \to & \mathsf{G}\ (1) \\ \mathsf{G}\ (2) & \to & 2 \end{array}$$

Reductions:

$$\begin{array}{lcl} \mathsf{F}\ (3,4) & \to_\perp & \perp \\ \mathsf{G}\ (1) & \to_\perp & \perp \\ \perp & \to_\perp & \perp \end{array}$$

3.4. LEMMA. (i) *Let $t \to_r s$. If $t' > t$ then there is an equivalent reduction $t' \to_r s'$, and $s' \geq s$.*

(ii) *Let $t \to_r s$. If $t' < t$ then there is an equivalent reduction $t' \to_r s'$ or $t' \to_\perp^\varepsilon s'$, and $s' \leq s$.*

(iii) *If $s < t$ and $t \to_\perp \perp$ then also $s \to_\perp \perp$.*

(iv) *$\to_\perp$ is confluent.*

PROOF. (i) Let $t = C[\Delta]$, such that Δ is the redex going to be rewritten. Then t' can be written as $C'[\Delta']$ such that $C \leq C'$ and $\Delta \leq \Delta'$. We can write Δ' as $C''[s_1, \ldots, s_n]$ such that $\Delta = C''[\perp^n]$. The n $\perp$'s don't contribute to the redex. Thus by lemma 2.6, Δ' will reduce to $C'''[s_1^{i_1}, \ldots, s_n^{i_n}]$, and Δ to $C'''[\perp^{i_1}, \ldots, \perp^{i_n}]$.

(ii) Let t/u be the redex going to be rewritten. If t'/u is not defined, or is $\bot$, then $t' < s$. Otherwise, consider the term t'/u. If it is a redex, it reduces by the same rule as t/u. Thus suppose it is no redex. We can write both terms as $t/u = C[s_1, \ldots, s_n]$ and $t'/u = C[\bot^n]$. At least one of the terms s_i contributes to the redex t/u. Now, by orthogonality, t'/u can never become a redex of another rule. Also, it cannot become a redex for the rule of the redex t/u: there would be a reduction to a term such that none of the residuals of s_i contributes to the redex (note that the contributing parts of a redex cannot change by reduction of subterms). Thus $t'/u \to_\bot \bot$.

(iii) Suppose s has a chnf. Then there is a reduction sequence $s \to_r s_1 \to_r \ldots \to_r s_n$ such that s_n is a chnf. Because of lemma 3.4.i there is an equivalent reduction sequence $t \to_r t_1 \to_r \ldots \to_r t_n$ such that $t_i \geq s_i$. However, then t_n is a chnf, thus, by contradiction, s has no chnf.

(iv) Let $t \in T_\bot$, containing two $\bot$-redexes t_1 and t_2. If they are disjoint ($t = C[t_1, t_2]$), there is a trivial common reduct ($C[\bot, \bot]$). Suppose $t_1 \subseteq t_2 = C[t_1]$. Then $C[t_1] \to_\bot C[\bot]$, and $C[t_1] \to_\bot \bot$. Because $C[\bot] \leq C[t_1]$ also $C[\bot] \to_\bot \bot$ (lemma 3.4.iii). Confluence of $\to_\bot$ follows directly. □

The first two parts of the above lemma state that related terms can 'simulate' each other reductions. Let $s, t \in T_\bot$ such that $s \leq t$. Now, if s/u is a redex, than also t/u is a redex. If contraction of these redexes gives us terms s' and t' then $s' \leq t'$. We say that the reduction in t simulates the reduction in s. The other way around is rather similar. Let t/u be a redex. If s/u is defined, this term is a redex (eventually with respect to $\to_\bot$). Otherwise, there is an occurrence v such that t/u is a subterm of t/v and $s/v = \bot$. Again, reductions of the terms at position u (or v) gives us terms s' and t' such that $s' \leq t'$.

3.5. LEMMA. *Let $t, t_r, t_\bot \in T_\bot$ such that $t \to_r t_r$ and $t \to_\bot t_\bot$. Then there is a term $t' \in T_\bot$ such that $t_r \twoheadrightarrow_\bot t'$ and $t_\bot \to_r^\varepsilon t'$.*

PROOF. Let Δ_r and $\Delta_\bot$ be the subterms of t being rewritten (such that $t = C_\bot[\Delta_\bot] = C_r[\Delta_r]$ and $\Delta_r \to_r s$). We have three cases:

(i) Δ_r and $\Delta_\bot$ are disjoint: trivial.

(ii) $\Delta_r \subseteq \Delta_\bot$. Thus $\Delta_\bot = C[\Delta_r]$, and $C[\Delta_r] \to_r C[s]$. Since $\Delta_\bot$ has no chnf than also $C[s]$ has no chnf. Thus $t = C_\bot[\Delta_\bot] \to_\bot C_\bot[\bot]$ and $t = C_\bot[C[\Delta_r]] \to_r C_\bot[C[s]] \to_\bot C_\bot[\bot]$.

(iii) $\Delta_\bot \subset \Delta_r$. Thus $\Delta_r = C[\Delta_\bot]$. $\Delta_\bot$ doesn't contribute to redex Δ_r. Thus, there is a context C' such that $s = C'[\Delta_\bot, \ldots, \Delta_\bot]$, and $C[\Delta_\bot] \to_\bot C[\bot] \to_r C'[\bot, \ldots, \bot]$. Clearly, $s \twoheadrightarrow_\bot C'[\bot, \ldots, \bot]$. □

3.6. PROPOSITION. *$\to_{\bot r}$ is confluent.*

PROOF. Because of lemma 3.5, $\twoheadrightarrow_\bot$ and $\twoheadrightarrow_r$ are commutative. Since $\to_\bot$ and $\to_r$ are both confluent (lemma 3.4 and theorem 2.4), it follows by the Hindley-Rosen lemma (see, for example, Barendregt [1984]) that $\to_{\bot r}$ is confluent. □

In the following, we will abbreviate $\to_{\bot r}$ simply as $\to$. We can now state when a function is strict in an argument.

3.7. DEFINITION. A function F is *strict* in its i-th argument if for all terms $t_1 \ldots t_{i-1}, t_{i+1} \ldots t_n \in T$:

$$F(t_1, \ldots, t_{i-1}, \bot, t_{i+1}, \ldots, t_n) \twoheadrightarrow \bot$$

Because of the confluence of $\to$, strictness is a transitivity property. That is, let t be a term having no chnf, and F be a function that is strict in its i-th argument. Then $F(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n) \twoheadrightarrow \bot$. However, this doesn't yet mean that a term on such a strict position can safely (or needs to) be reduced to constructor head normal form. For proving that this notion of strictness corresponds to the intention of strictness we need a relationship between $\bot$-reduction and neededness. This is the topic of the next section.

4. Neededness: redexes and normalisation

Normally, neededness is related to normal forms. We will define a similar notion for the constructor head normal form case. We start by considering standard reduction in T.

4.1. DEFINITION. A redex Δ in a term $t = C[\Delta]$ is a *ch-needed* redex if in every reduction sequence that leads to a chnf Δ or one of its residuals is reduced.

If a term has no chnf all its redexes are ch-needed. It is a well-known fact that reduction of needed redexes leads to the normal form (if existing). A similar property can be derived for the chnf case. First, we recall some basic notions.

4.2. DEFINITION. (i) Given a TRS (Σ, R), a *strategy* S for this system is a map which assigns to each element t of $T(\Sigma)$ a possibly empty set of reduction sequences $S(t)$, all starting with t. A strategy defines a new reduction relation: $t \to_s s$ if $t \twoheadrightarrow s \in S(t)$. We speak of a *one-step* reduction strategy if each element of $S(t)$ has length 1.

(ii) A strategy S is *constructor head normalising* if for all terms t having a constructor head normal form, $t \to_s t_1 \to_s \ldots$ must eventually terminate with a constructor head normal form (that is, each reduction sequence $\to_s$ starting at a term having a constructor head normal form is finite).

(iii) Let S be a strategy. The strategy *quasi-S* is defined as follows: $\mathit{quasi}\text{-}S(t) = \{t \twoheadrightarrow t' \to_s s\}$. Thus a *quasi-S* reduction sequence is a S reduction sequence, diluted with arbitrary reduction steps.

(iv) A strategy S is *constructor head hypernormalising* if *quasi-S* is constructor head normalising.

4.3. THEOREM. (i) *Every term t, not in chnf, contains a ch-needed redex.*

(ii) *Repeated contraction of ch-needed redexes is constructor head (hyper)normalising.*

PROOF. The proof is quite a lot of work. It is comparable to the proof for the normal form case, only some minor, but straightforward changes are necessary. We refer to Kennaway & Sleep [1989] or Huet & Lévy [1979]. □

In the sequel, we will just speak of needed(ness) instead of ch-needed(ness). Similarly, we will speak of normalising instead of constructor head normalising.

We return now our attention to terms from $T_\perp$. We extend the notion of neededness in the obvious way to terms of $T_\perp$ and the reduction relation $\to$. We start by relating neededness and $\perp$-reduction.

4.4. LEMMA. *A non-needed redex cannot contain a needed redex.*

PROOF. The proof is based on the fact that reductions inside a redex have no influence on reductions outside that redex. Thus, let $t_0 = C[\Delta]$ with Δ a non-needed redex. Let $t_0 \to t_1 \to \ldots t_n$ be a reduction sequence to chnf, in which no residual of Δ is reduced. That is, we can write each t_i as $C_i[\Delta_{i_1}, \ldots, \Delta_{i_n}]$ where Δ_{i_j} are the residuals (all still being redexes of the same rule). Each reduction step changes either C_i (by an outer reduction), or one of the residuals (by an inside reduction). By application of lemma 2.6 we can skip the inside reductions, obtaining a reduction to $C_n[\ldots]$. This last term is in chnf. Thus none of the terms inside Δ is needed. □

4.5. PROPOSITION. *Let $t = C[\Delta]$ with Δ a redex. Then*

$$\Delta \textit{ needed} \Leftrightarrow C[\perp] \to_\perp \perp$$

PROOF. ($\Rightarrow$) Let Δ be needed. Suppose $C[\perp] \twoheadrightarrow_r t_c$, with t_c a chnf. Then, by lemma 3.4, there is an equivalent reduction sequence $C[\Delta] \twoheadrightarrow_r t'_c$, such that $t_c \leq t'_c$. In this reduction sequence Δ, nor one of its subterms or residuals, is reduced (this follows from inspection of the proof of lemma 3.4) (contradiction). Thus $C[\perp] \to_\perp \perp$. ($\Leftarrow$) Let $C[\perp] \to_\perp \perp$. Suppose $C[\Delta] \twoheadrightarrow_r t_c$ and Δ not needed (thus $t_c \neq \perp$). Every subredex of Δ is not needed (lemma 4.4), Then there is a reduction sequence in which no residual of Δ (nor one of its subterms) is reduced. In this sequence, none of the residuals of Δ did contribute to a reduction. Then, by repeated application of lemma 2.6, the same reduction sequence exists for $C[\perp]$, leading to a chnf (contradiction). □

We also want to relate neededness and needed reductions in related terms, i.e. terms t and t' for which $t \leq t'$. This is done in the following proposition.

4.6. LEMMA. *A residual of a non-needed redex is non-needed.*

PROOF. Simple application of proposition 4.5 and lemmas 2.6 and 3.4. □

4.7. PROPOSITION. (i) *Let $t, t' \in T_\perp$ such that $t \leq t'$. Suppose that t has a chnf. Then*

$$t'/u \text{ needed in } t' \Leftrightarrow t/u \text{ needed in } t$$

(ii) *Let $t \to s$. Let $t' \leq t$ such that t' has a chnf. Then there is a $s' \leq s$ such that $t' \to^\varepsilon s'$. Moreover, if the reduction step was a needed reduction step, then $t' \to_r s'$ and also this reduction step is a needed one.*

PROOF. (i) ($\Rightarrow$) Let $t' = C'[\Delta']$ such that $\Delta' = t'/u$. We have $C'[\perp] \twoheadrightarrow \perp$. If t/u is undefined, then $t \leq C'[\perp]$ and t has no chnf. Thus, let $t = C[\Delta]$ such that $\Delta = t/u$. Clearly, $\Delta \leq \Delta'$ and $C \leq C'$. Thus if $C'[\perp] \twoheadrightarrow \perp$ then also $C[\perp] \twoheadrightarrow \perp$ (lemma 3.4). ($\Leftarrow$) Let $\Delta = t/u$ the needed redex in t, and $\Delta' = t'/u$. There is a common context C_0 such that $t = C_0[\perp, \ldots, \perp, \Delta]$ and $t' = C_0[s_1, \ldots, s_n, \Delta']$. Now, none of the s_i is on a needed position (otherwise, by the other direction of this proposition, there would be a $\perp$ on a needed position in t). Now, we claim that then Δ' is needed. We proof this as follows. Let, without loss of generality, all the s_i be non-needed redexes. Suppose that Δ' is not needed. Then, there is a reduction to chnf such that all reductions take place in C_0 (note that, because of lemma 4.6, all the non-needed redexes stay non-needed). However, this would mean that also Δ in t is not needed, since a similar reduction exists. Thus Δ' must be a needed redex.

(ii) This follows immediately from (i) and lemma 3.4.ii. □

4.8. REMARK. Theorem 4.3 holds for terms in T. It is easy to extend the theorem for terms from $T_\perp$.

PROOF. Let $t \in T_\perp$ and $t' \in T$ such that $t \leq t'$. Suppose t has a chnf.

(i) t' has a needed redex, and thus by proposition 4.7.i also t.

(ii) A needed redex in t is also a needed redex in t' (proposition 4.7.i). If there is an infinite reduction sequence for t, contracting enough needed redexes, there is also an equivalent infinite reduction sequence for t', contracting similar needed redexes. □

We will say that a needed redex in a term is on a *needed* or *strict position*. Clearly, a term with a subterm having no chnf on a needed position has no chnf. A way of determining needed positions is by using strictness information. Proposition 4.5 clearly shows that our definition of strictness corresponds with the required intention of strictness.

We continue now with the definition of the needed reduction we want to use for abstract reductions.

4.9. DEFINITION. (i) The needed reduction relation $\to_n$ is defined as follows: $t \to_n t'$ if $t \twoheadrightarrow t'$ and at least one needed reduction is done.

(ii) A subterm s in a term $t = C[s]$ is a *needed subterm* if it is on a needed position, i.e. if $C[\perp] \to_\perp \perp$.

Because of theorem 4.3, $\to_n$ is chnf-normalising (but not necessarily confluent: it might be possible to reduce a term to two different chnf's). We note that $\to_n^+ = \to_n$, that is, $\to_n$ is transitive. Needed subterms have a lot of nice properties.

4.10. PROPOSITION (Some useful properties of needed subterms). *Let $t \in T_\perp$ be a term having a chnf.*

(i) *Each needed subterm of t, not being in chnf, contains a needed redex. The subterm has a chnf.*

(ii) *Reduction of needed redexes within a needed subterm of t leads to the chnf of the subterm, and, by repeated application, of t itself.*

(iii) *Each needed subterm of t, not being in chnf, has a descendant that is a needed redex.*

(iv) *A non-needed subterm cannot contain a needed subterm.*

(v) *Let $t = C[s]$ such that s is a non-needed subterm. Then $C[\bot]$ has a chnf.*

(vi) *t can be written as $t = C[s_1, \ldots, s_n]$ such that all subterms s_i are non-needed, and all subterms in C are needed. The term $C[\bot^n]$ has a chnf.*

PROOF. (i) Let $t = C[s]$, such that s is a needed subterm. This subterm must have a chnf (otherwise, t also has none). Clearly, s contains a needed redex, such that $s = C'[\Delta]$ and $C'[\bot] \to \bot$. Then also $C[C'[\bot]] \to C[\bot] \to \bot$. Thus Δ is needed in t also.

(ii) Simple application of theorem 4.3 and (i).

(iii) This follows from the fact that each term, not being in chnf but having a chnf, must eventually become a redex.

(iv) Let $t = C[s]$, with s a non-needed subterm. Suppose s contains a needed subterm, i.e. let $s = C'[s']$ such that s' is a needed subterm. Thus, $C[C'[\bot]]$ has no chnf. Then, also $C[\bot]$ has no chnf (contradiction). Thus any subterm of s is non-needed.

(v) Trivial.

(vi) That t can be written as such follows immediately from (iv). The statement follows by repeated application of (v), and the fact that the neededness properties of the subterms are not influenced by this operation (proposition 4.7.i). □

Proposition 4.10.i and 4.10.ii state that if a needed redex (or subterm) has been found, this subterm can safely be reduced to chnf by looking for needed redexes (and subterms) within the subterm itself. Proposition 4.10.iv-vi state that we can remove non-needed parts of a term without influencing the chnf property of that term. The needed parts of a term are of special interest:

4.11. DEFINITION. Let $t = C[s_1, \ldots, s_n]$ such that all subterms s_i are non-needed, and all subterms in C are needed. The *needed part* of t is $\mathcal{N}(t) = C[\bot^n]$.

5. Approximations

In this section we will introduce the notion of *approximations*. These approximations will be used for describing the basic machinery of the abstract reduction mechanism. To some extent, terms defined by an approximation (*abstract* terms) can be considered as representations of sets of elements of $T_\bot$ (*concrete* terms).

5.1. DEFINITION (Approximations). An *approximation* $\mathcal{A}$ of the TRS (Σ, R) contains:

(i) A set of terms $T_\alpha = T(\Sigma \cup \{\bot, \top\} \cup U)$ where $U = \{U_i \mid i \geq 1\}$. We suppose that the new symbols are not contained in Σ. A symbol U_i, called a union symbol, has arity i. For convenience, we will write $\langle t_1, \ldots, t_n \rangle$ instead of $U_n(t_1, \ldots, t_n)$.

(ii) A preordering $\leq_\alpha$ defined as follows:

$\bot$	$\leq_\alpha$	s	for all $s \in T_\alpha$.
t	$\leq_\alpha$	$\top$	for all $t \in T_\alpha$.
$F(t_1, \ldots, t_n)$	$\leq_\alpha$	$F(s_1, \ldots, s_n)$	$(n \geq 0)$ if $t_i \leq_\alpha s_i$ for all $i = 1 \ldots n$.
$\langle t_1, \ldots, t_n \rangle$	$\leq_\alpha$	s	$(n \geq 1)$ if $t_i \leq_\alpha s$ for all $i = 1 \ldots n$.
t	$\leq_\alpha$	$\langle s_1, \ldots, s_n \rangle$	$(n \geq 1)$ if $t \leq_\alpha s_i$ for some $i = 1 \ldots n$.

(iii) A reduction relation $\to_\alpha$ defined as follows (let $t_\alpha, t'_\alpha \in T_\alpha$ and $t, t' \in T_\perp$):

$$t_\alpha \to_\alpha t'_\alpha \Leftrightarrow \forall t \leq t_\alpha\, [\exists t' \leq t'_\alpha\, [t \to_n t']]$$

A term t_α for which there is no reduction $t_\alpha \to_\alpha t'_\alpha$ is in chnf.

We consider T_α as an extension of $T_\perp$, but it is also possible to consider T_α as a new entity. For convenience, we define a concretisation function $\gamma\colon T_\alpha \to \mathcal{P}(T_\perp)$ as follows: $\gamma[t_\alpha] = \{t \in T_\perp \mid t \leq_\alpha t_\alpha\}$. Clearly, $\leq$ from $T_\perp$ is included in $\leq_\alpha$. We will omit the subscript α if no confusion can occur.

REMARK. The ordering induces an equivalence relation over T_α. We write $t \equiv s$ if $t \leq_\alpha s$ and $s \leq_\alpha t$. For any term t, let $t^\equiv$ be the equivalence class belonging to t. The ordering $\leq_\alpha$ is extended to equivalence classes as follows: $t^\equiv \preceq s^\equiv \Leftrightarrow \exists t \in t^\equiv\, [\exists s \in s^\equiv\, [t \leq_\alpha s]]$. We note that $(T_\alpha^\equiv, \preceq)$ is a lattice but *not* a complete lattice (see Goubault & Hankin [1991]).

5.2. LEMMA (Equivalences). *We have (amongst others) the following equivalences (let $t, t_1 \ldots t_n \in T_\alpha$):*

$\langle t, t_1, \ldots, t_n\rangle$	$\equiv$	$\langle t_1, \ldots, t_n\rangle$	*if $t \leq t_i$ for some $1 \leq i \leq n$*
$C[\langle t_1, \ldots, t_n\rangle]$	$\equiv$	$\langle C[t_1], \ldots, C[t_n]\rangle$	*for any context $C[]$*
$\langle t\rangle$	$\equiv$	t	
$\langle t_1, \ldots, t_n\rangle$	$\equiv$	$\langle \pi(t_1, \ldots, t_n)\rangle$	*for any permutation π*
$\langle \top, t_1, \ldots, t_n\rangle$	$\equiv$	$\top$	
$\langle \perp, t_1, \ldots, t_n\rangle$	$\equiv$	$\langle t_1, \ldots, t_n\rangle$	

PROOF. Straightforward. □

The symbol $\top$ represents all terms in $T_\perp$. The union symbols are used to combine abstract values (or sets). For convenience, we allow unions of any arity, but it would have been sufficient to allow only unions with arity 2.

The abstract reduction relation is defined in terms of the concrete needed reduction relation (note that the reduction $\to_n$ is included in $\to_\alpha$). In words it means that $t_\alpha \to_\alpha t'_\alpha$ if the reduction "simulates" at least one needed reduction for each term in t_α. At this point, $\to_\alpha$ is just a formal definition. We have not yet specified how to *compute* such abstract reduction steps (and, of course, $\to_\alpha$ is not completely computable). In the following, we will define various possibilities to determine abstract reduction steps (i.e. lemma 5.4 and 5.9, prop. 5.10 and section 6).

We start by expressing the strictness test in the abstract world:

5.3. PROPOSITION (Strictness test). *Let $F(\top, \ldots, \top, \perp_i, \top, \ldots, \top) \twoheadrightarrow_\alpha \perp$ (where the subscript i denotes the i-th argument of F). Then the function F is strict in its i-th argument.*

PROOF. Consider a term $t = F(t_1, \ldots, t_{i-1}, \perp, t_{i+1}, \ldots, t_n)$ for some terms $t_{j, j\neq i} \in T_\perp$. Since $t \leq F(\top, \ldots, \top, \perp, \top, \ldots, \top)$, there is a reduction $t \twoheadrightarrow_n \perp$. □

The abstract reduction relation itself already has some nice properties.

5.4. LEMMA. (i) *For any term $t \in T_\alpha$, not in chnf, we have $t \to_\alpha \top$ (Top-introduction). However, $\to_\alpha$ is not necessarily confluent.*

(ii) *If $t \to_\alpha \cdot \to_\alpha t'$ then $t \to_\alpha t'$ (thus $\to_\alpha^+ = \to_\alpha$).*

(iii) *Let $t = C[s]$ and $t' = C[s']$ such that $s \to_\alpha s'$. If $t' \to_\alpha t''$ then $t \to_\alpha t''$.*

PROOF. (i) Trivial. Confluence is not guaranteed since it is possible that $t_\alpha \to_\alpha t'_\alpha$ and $t_\alpha \to_\alpha t''_\alpha$ such that t'_α as well as t''_α are in chnf but $t'_\alpha \neq t''_\alpha$.

(ii-iii) Straightforward. □

The last three properties of α-reduction are important. The first one states that we can introduce approximations (for example to ensure termination in an implementation). The 'transitivity' property of the reduction will be appear to be useful for many other properties. The last part of the lemma states that we can safely do additional reductions in subterms if necessary. This implies that sharing of subterms is safe, which is important for an implementation of abstract reduction.

As in $T_\perp$ we can define a notion of neededness. Informally, a subterm of $t_\alpha \in T_\alpha$ is needed if all its corresponding subterms in all terms t_α represents are needed. To formalise the notion of corresponding subterms, we need a precise relationship between subterms of an abstract term and subterms of the concrete terms it represents. This is done by the notion of *embeddings*. An embedding acts as a kind of homomorphism. Let $t \in \gamma[t_\alpha]$. An embedding maps the structure of t on t_α. Each subterm of t is mapped on a corresponding subterm of t_α, not being a union. Given a subterm of t_α, is not necessarily the case that t has a corresponding subterm (because of the presence of unions). Therefore, we will consider a *set* of needed subterms of t_α to define neededness.

5.5. DEFINITION. Let $\varphi: O(t) \to O(t_\alpha)$ be an occurrence mapping. This mapping is an *embedding* if for all $(u_1 \dots u_m) \in O(t)$:

(i) $root(t/u_1 \dots u_m) \leq root(t_\alpha/\varphi(u_1 \dots u_m))$ (where $\perp \leq F \leq \top$ for any symbol F)

(ii) $\varphi(u_1 \dots u_m) = \varphi(u_1 \dots u_{m-1})$ if $root(t_\alpha/\varphi(u_1 \dots u_{m-1})) = \top$, and otherwise
$\varphi(u_1 \dots u_m) = \varphi(u_1 \dots u_{m-1}).u_m.v_1 \dots v_n$ such that $t_\alpha/\varphi(u_1 \dots u_{m-1}).u_m.v_1 \dots v_i$ is a union node (for all $i < n$).

(iii) $\varphi(\bullet) = v_1 \dots v_n$ such that $t_\alpha/v_1 \dots v_i$ is a union node (for $i < n$).

5.6. LEMMA. *Let $t \in \gamma[t_\alpha]$. Then there exists an embedding $\varphi: O(t) \to O(t_\alpha)$.*

PROOF. We can construct inductively an embedding as follows:

(i) if $t_\alpha = \top$, or $t_\alpha = \perp$. Let $\varphi(u) = \bullet$ for all $u \in O(t)$.

(ii) if $t_\alpha = \langle t_1, \dots, t_n \rangle$. Then there is an $1 \leq i \leq n$ such that $t \in \gamma[t_i]$. We define for all $u \in O(t)$: $\varphi(u) = i.\varphi'(u)$ where $\varphi': O(t) \to O(t_i)$ is an embedding.

(iii) if $t_\alpha = F(t'_1, \dots, t'_n)$. Then $t = F(t_1, \dots, t_n)$ or $t = \perp$. The last case is trivial. Otherwise, We define $\varphi(\bullet) = \bullet$ and for all $(i.u) \in O(t)$: $\varphi(i.u) = i.\varphi_i(u)$ where $\varphi_i: O(t_i) \to O(t'_i)$ are embeddings (for $1 \leq i \leq n$). □

The following lemma gives some useful properties of embeddings.

5.7. LEMMA. *Let $t \in \gamma[t_\alpha]$ and $\varphi: O(t) \to O(t_\alpha)$ an embedding.*

(i) *$t/u \in \gamma[t_\alpha/\varphi(u)]$ for all $u \in O(t)$.*

(ii) *Suppose $t_\alpha = C[\perp]$. Then $t \in \gamma[C[s]]$ for any $s \in T_\alpha$ and φ is also an embedding for $C[s]$.*

(iii) *Let $t_\alpha = C[s]$, and s no union, and v the occurrence of s. If there is no $u \in O(t)$ such that $\varphi(u) = v$, then $t \in \gamma[C[\perp]]$ and $\varphi: O(t) \to O(C[\perp])$ is also an embedding.*

PROOF. (i) By induction to the definition (or construction) of φ.

(ii) Clearly, $t \leq C[s]$. Let v be the occurrence of s or $\perp$ in t_α. If there is a subterm t/u of t such that $\varphi(u) = v$, then $t/u = \perp$.

(iii) It is easy to show that all subterms are mapped to terms outside s. That is, there is no $u \in O(t)$ such that $\varphi(u) = v.v_1 \dots v_n$ for any sequence $v_1 \dots v_n$. □

5.8. DEFINITION (Neededness). Let $S = \{s_1, \dots, s_n\}$ be a non-empty set of disjoint subterms of a term t_α, such that $s_i = t_\alpha/u_i$ for $1 \leq i \leq n$. S is *needed* if for all $t \in \gamma[t_\alpha]$ there is a $t' \in \gamma[t_\alpha])$ such that $t \leq t'$ and such that there is a corresponding needed subterm $s = t'/u$, and $u_i = \varphi(u)$ for some $1 \leq i \leq n$, and any embedding $\varphi: O(t') \to O(t_\alpha)$.

Suppose, for example, a term $\mathsf{t}_\alpha = \langle +\ (\mathsf{s},1),-\ (\mathsf{s}',2)\rangle$, where + and - are the usual strict integer operations. Then $\{\mathsf{s},\mathsf{s}'\}$ is a set of needed subterms, but also $\{+\ (\mathsf{s},1),-\ (\mathsf{s}',2)\}$ is such a set. Any concrete term represented by t_α contains a needed subterm, corresponding to one of the elements of the set. The t' in the definition of neededness is necessary because not every term has a subterm directly mapping on one of the elements of the set (e.g. $\bot$ and the set $\{\mathsf{s},\mathsf{s}'\}$). In such a case, the subterm of t' corresponds with a $\bot$ subterm of t.

5.9. LEMMA. *Let $t_\alpha \in T_\alpha$. Let $\{s_1,\ldots,s_n\}$ be a set of needed subterms in t_α such that $t_\alpha = C[s_1,\ldots,s_n]$.*

(i) *Neededness in T_α is directly related to neededness in $T_\bot$. That is, for all $t \in \gamma[C[\bot^n]]$ it holds that $t \to_\bot \bot$ (that is: $C[\bot^n] \to_\alpha \bot$).*

(ii) *Let $s_i \to_\alpha s'_i$ for $1 \le i \le n$. Then $t_\alpha \to_\alpha C[s'_1,\ldots,s'_n]$.*

PROOF. (i) Let $t \le t'_\alpha = C[\bot^n]$. Let φ be an embedding from $O(t)$ to $O(t'_\alpha)$. Then, φ is also an embedding from $O(t)$ to $O(t_\alpha)$ (lemma 5.7). Let $s = t/u$ be the needed subterm such that $s_i = t_\alpha/\varphi(u)$. Since $t'_\alpha/\varphi(u) = \bot$, it follows that $s = \bot$, and thus $t \to_\bot \bot$.

(ii) Let $t \le t_\alpha$. t has some disjoint needed subterms $t_1 \ldots t_m$, at least one, such that for all $1 \le j \le m$, $t_j = t/u_j$ and $t_\alpha/\varphi(u_j) = s_i$ for some $1 \le i \le n$. Let $t_j \to_n r_j$ such that $r_j \le s_i$. Let $t = C[t_1,\ldots,t_m]$. Then $t \to_n C[r_1,\ldots,r_m] \le C[s'_1,\ldots,s'_n]$. □

The second part of the lemma states that reduction of a set of needed subterms is an α-reduction step. This property is very useful in an implementation of abstract reduction. We are now ready for the proof of the correctness of $\bot$-introduction, as described in the introduction.

5.10. PROPOSITION (Bottom introduction). *Let $t_\alpha \to_\alpha t'_\alpha$ such that t'_α has a needed set of subterms $\{s_1 \ldots s_k\}$. Furthermore, let $s_i \le t_\alpha$ for $1 \le i \le k$. Then $t_\alpha \to_\alpha \bot$.*

PROOF. We can construct a reduction sequence of needed reductions, leading to $\bot$ or to infinity, as follows. First we note that t'_α contains no chnfs. Let $t \in \gamma[t_\alpha]$. Then there is a reduction sequence such that $t \to_n t'$ and $t' \in \gamma[t'_\alpha]$. If $t' = \bot$, we are ready. Otherwise, t' contains a subterm $s = t'/u$ that is needed, and such that $s \le t'_\alpha/\varphi(u) = t'_\alpha/u_i \le t_\alpha$. Note that s is not in chnf. The needed reduction sequence can be continued by reducing s to chnf (proposition 4.10). Since $s \in \gamma[t_\alpha]$, we can repeat the argument for s. Now we have a needed reduction sequence, which terminates with $\bot$, or is infinite. Since $\to_n$ is hypernormalising t cannot have a chnf. □

5.11. COROLLARY. *Let $t_\alpha \to_\alpha^+ t'_\alpha$ such that $t'_\alpha \le t_\alpha$. Then: $t_\alpha \to_\alpha \bot$.*

PROOF. For all $t \in \gamma[t'_\alpha]$ it holds that t is needed. Thus $\{t'_\alpha\}$ is a needed set of subterms. □

6. τ-Reduction

In the previous section, we described only some (be it important) properties of abstract reduction. Now, we will take a more operational viewpoint. We start by defining τ-reduction. It describes how a term might rewrite, based on the possible rewritings in the orthogonal term rewriting system itself. Combining all possible τ-reduction steps for a term gives us certain α-reduction steps for that term. Given a term not in chnf, we can derive α-reduction steps, but we cannot necessarily determine all, or the most optimal α-reduction steps. This form of τ-reduction can form the basis of a concrete implementation of abstract reduction. For an example of this, we refer to Nöcker [1993].

6.1. DEFINITION (τ-reduction). (i) A term $t \in T_\alpha$ not in chnf is a τ-redex if there is a term $t' \in \gamma[t]$ and t' is a redex with respect to $\to$ (so note that $\top$ is no τ-redex).

(ii) Let $t' \in T_\alpha$ such that $t' = l^\sigma$, $t' \le t$ for some term l and a substitution σ. The substitution σ is *maximal* if there is no substitution σ' such that $t' < \sigma'(l) \le t$.

(iii) The reduction relation $\to_\tau$ over T_α is defined as follows (consider a term $C[t]$ with t a τ-redex):

(a) $C[t] \to_\tau C[r^\sigma]$ if there is a rewrite rule **f**: $r \to l$ such that $l^\sigma \leq t$ and σ maximal.

(b) $C[t] \to_\tau C[\bot]$ otherwise

Intuitively spoken, a term is a τ-redex if it 'matches' a left-hand side. Matching is the same as for ordinary term rewriting, except for the term $\top$ which matches every pattern, and for Unions. A substitution should be maximal in order to simulate as many as possible reductions. Maximal substitutions can easily by found by pattern matching. Consider as an example the rewrite rule:

$$\mathsf{F}\ (\mathsf{A}\ (\mathsf{x}),\ 1,\ 1) \quad \to \quad \mathsf{B}\ (\mathsf{x})$$

The term $\mathsf{F}\ (\top,\ \top,\ 1)$ matches the left-hand side, and will rewrite to $\mathsf{B}\ (\top)$. That is, the variable x is bound to $\top$. Note that the subterms 1 of the left-hand side correspond with two different terms in the redex. This means that the redex cannot be written as a direct substitution of the left-hand side (i.e. there is no σ such that $\sigma\ (\mathsf{F}\ (\mathsf{A}\ (\mathsf{x}),\ 1,\ 1)) = \mathsf{F}\ (\top,\ \top,\ 1)$). Similarly, the term $\mathsf{F}\ (\langle \mathsf{A}\ (2),\ \mathsf{A}\ (3)\rangle, 1,\ 1)$ will rewrite to $\mathsf{B}\ (\langle 2,\ 3\rangle)$, that is, x is bound to $\langle 2,\ 3\rangle$.

6.2. Lemma. *Let $t \in T_\alpha$ be a τ-redex. If there is a term $t' \in \gamma[t]$ and t' is a redex with respect to $\to_r$, then there is a maximal substitution.*

Proof. A maximal substitution can be found by a pattern matching algorithm. We define informally a matching process $Match(p, t)$ for a pattern p and a term t as follows:

if	$p = x$	Define $\sigma(x) = t$
if	$p = F(p_1, \ldots, p_n)$	
	if $t = \top$	Do for all $1 \leq i \leq n$ $Match(p_i, t)$
	if $t = F(t_1, \ldots, t_n)$	Do for all $1 \leq i \leq n$ $Match(p_i, t_i)$
	if $t = \langle t_1, \ldots, t_n \rangle$	Do for all $1 \leq i \leq n$ $Match(p, t_i)$ for which $p \leq t_i$

This process defines a substitution σ for all variables in p. Since a variable might be reached several times, the union of all its definitions has to be taken. By induction to the structure of p we can proof that $\sigma(p) \leq t$ and that σ is maximal. □

A τ-reduction in a term simulates all possible reductions of all corresponding subterms in the concrete terms (with respect to one rule). If a τ-redex has no maximal substitution, it reduces to $\bot$. A term can have (finitely) many τ-redexes. A term can even be a τ-redex for different rewrite rules. To simulate the necessary needed reductions it is sufficient to reduce all τ-redexes, and to take the union of all the results. This form of α-reduction is expressed in the following proposition.

6.3. Lemma. *Let $t \in T_\alpha$ be a τ-redex.*

(i) *Let $\Delta \in \gamma[t]$ be a redex. If $\Delta \to r$, and $t \to_\tau t'$. Then $r \in \gamma[t']$.*

(ii) *Let $s \in T_\alpha$ such that $t \to_\tau s$. If $t' \geq t$ then $t' \to_\tau s'$ and $s' \geq s$ for some $s' \in T_\alpha$.*

Proof. Straightforward. □

6.4. Proposition. *Consider a term $t \in T_\alpha$. Let t have n different τ-redexes Δ_i, such that $t = C_i[\Delta_i]$ and $C_i[\Delta_i] \to_\tau t_i$ for $i = 1 \ldots n$. Then $t \to_\alpha \langle t_1, \ldots, t_n \rangle$.*

Proof. First, we note that t has a finite number of τ-redexes, since we assumed that functions are finite. Thus $\langle t_1, \ldots, t_n \rangle$ is a finite term. Consider now a term $t' \in \gamma[t]$, and φ an embedding. Let t' have a chnf, then t' has a needed redex. Let $t' = C[\Delta]$, with $\Delta = t'/u$ the needed redex. Then, $\Delta \leq t/\varphi(u)$. Now, by lemma 6.3, $t/\varphi(u)$ is a τ-redex (say Δ_1). Let $\Delta \to r$. Then $C[r] \leq C_1[t_1] \leq \langle t_1, \ldots, t_n \rangle$. Thus all needed reductions are simulated. □

The reduction $\to_\alpha$ as constructed by proposition 6.4 can be used as the basis for an implementation. Termination can be assured by Top-introduction (lemma 5.4.i). However, this reduction system is not efficient, and it is very difficult to decide what parts of a term have to be reduced.

The efficiency can be improved in various ways. It is possible to apply the properties of α-reduction as derived in the previous section. For example, it is possible to use strictness information derived earlier (lemma 5.9.ii). Also $\bot$-introduction can be applied if possible. However, it is also possible to optimise the basic form of τ-reduction by simulating less reductions. For example, it is sufficient to simulate reductions of outermost needed redexes only. Such extensions can become rather complex. Here, we note that also specific properties of the functional language can be taken into account. For example, typing or the reduction strategy can introduce additional sources for optimisations. For a detailed discussion of this topic we refer to Nöcker [1993].

Consider for an example of abstract reductions and τ-reduction the following functions:

$$\begin{array}{lcl} \mathsf{F}\ (\mathsf{x},\mathsf{0}) & \to & \mathsf{x} \\ \mathsf{F}\ (\mathsf{x},\mathsf{1}) & \to & \mathsf{F}\ (\mathsf{x},\ \mathsf{G}\ (\mathsf{x})) \\ \\ \mathsf{G}\ (\mathsf{x}) & \to & \ldots \end{array}$$

(for any right-hand side of G). It is easy to show that F is strict in its second argument (since there are no maximal substitutions for $\mathsf{F}\ (\top,\bot)$). However, F is also strict in its first argument. This can be derived by analysing abstract reductions of $\mathsf{F}\ (\bot,\top)$:

$$\begin{array}{lcll} \mathsf{F}\ (\bot,\top) & \to_\alpha & \langle \bot, \mathsf{F}\ (\bot, \mathsf{G}\ (\bot))\rangle & (\tau\text{-reduction}) \\ & \equiv & \mathsf{F}\ (\bot, \mathsf{G}\ (\bot)) & \\ & \to_\alpha & \bot & (\bot\text{-introduction}) \end{array}$$

7. μ-Approximations

In this section we will extend the notion of approximation further. Consider as an introductory example the following rewrite rule:

$$\mathsf{F}\ (\mathsf{x}) \quad \to \quad \mathsf{A}\ (\mathsf{F}\ (\mathsf{x}))$$

where A is a constructor. We have the following abstract reduction step: $\mathsf{F}\ (\top) \to_\alpha \mathsf{A}\ (\mathsf{F}\ (\top))$. However, we would *like* to have a reduction $\mathsf{F}\ (\top) \to_\alpha \mu\mathsf{x}.\mathsf{A}\ (\mathsf{x})$, where the resulting term is a representation (a kind of cyclic term) for the infinite term $\mathsf{A}\ (\mathsf{A}\ (\mathsf{A}\ (\ldots)))$. However, the basic property for α-reduction doesn't hold, since any term resulting from a reduction of $\mathsf{F}\ (\mathsf{t})$, will contain the symbol F. We will solve this by introducing infinite terms and infinite reduction sequences, following the theory of transfinite reductions of Kennaway *et al.* [1990]. The goal is the definition of an extension of $\to_n$, such that it includes possibly infinite reduction sequences.

The next section briefly recalls some basic notions of transfinite reductions in orthogonal term rewriting systems. Then, we consider how constructor head normal forms behave in this context. In section 7.3 we introduce an extension of approximations based on the theory of transfinite reductions.

7.1. Basic notions of transfinite reductions in orthogonal trs's

We adopt the notions and notations of Kennaway *et al.* [1990]. Let $T(\Sigma)$ be a set of finite terms over an alphabet Σ. Its extension to a set of infinite terms is denoted as $T^\infty(\Sigma)$, or shortly T^∞. We suppose all left and right-hand sides of the rewrite rules to be finite. All important properties and notions of section 1 and 2 extend verbatim to infinite terms. Confluence of $\to_r$ is guaranteed only as long as reduction sequences are finite (or if some other conditions are satisfied) (Kennaway *et al.* [1990]). However, we will show that $\to_{\bot r}$ is confluent in all circumstances.

To discuss infinite (reduction) sequences and convergence, the following construction is made. The set of infinite terms is isomorphic to a complete metric space (Arnold & Nivat [1980]). Essential in the construction is the following notion of distance between terms of T^∞.

7.1. DEFINITION. The *distance* $d(s,t)$ between two terms s and t is defined as follows: it is zero if s and t are equal, 2^{-k} otherwise, where k is the largest depth up to where the terms are equal. Formally:

$$\begin{array}{lcll} d(F,F) & = & 0 & \text{for any nullary symbol } F \\ d(F(t_1,\ldots,t_n),F(s_1,\ldots,s_n)) & = & m/2 & \text{where } m = max(d(t_i,s_i)) \\ d(F(t_1,\ldots,t_n),G(s_1,\ldots,s_m)) & = & 1 & \text{otherwise} \end{array}$$

As a consequence, all converging sequences of length ω in T^∞ have a limit in T^∞. This can be generalised to (reduction) sequences of arbitrary ordinal length α.

7.2. DEFINITION. (i) A *sequence* of length α is a set of elements indexed by some ordinal $\alpha \geq 1$, notation, $(t_\beta)_{\beta<\alpha}$. Instead of $(t_\beta)_{\beta<\alpha+1}$ we can write $(t_\beta)_{\beta\leq\alpha}$.

(ii) A *reduction sequence* of length $\alpha \geq 1$ is a sequence $(t_\beta)_{\beta<\alpha}$ such that $t_\beta \to t_{\beta+1}$ for all $\beta+1<\alpha$. The depth of the redex contracted in t_β will be denoted as d_β.

7.3. DEFINITION (Convergence). By induction to the ordinal α we define when a sequence $(t_\beta)_{\beta\leq\alpha}$ is *converging* towards its limit t_α (notation $t_0 \to^c_\alpha t_\alpha$):

(i) $t_0 \to^c_0 t_0$.

(ii) $t_0 \to^c_{\beta+1} t_{\beta+1}$ if $t_0 \to^c_\beta t_\beta$.

(iii) $t_0 \to^c_\lambda t_\lambda$ if $t_0 \to^c_\beta t_\beta$ for all $\beta<\lambda$ and $\forall\varepsilon>0\,\exists\beta<\lambda\,\forall\gamma\,[\beta<\gamma<\lambda \Rightarrow d(t_\gamma,t_\lambda)<\varepsilon]$.

We now define *strongly converging* reduction sequences. In a strong reduction sequence the depth of the contracted redexes tends to infinity.

7.4. DEFINITION (Strong reduction). By induction to the ordinal α we define when a reduction sequence is a *strong* reduction:

(i) $(t_\beta)_{\beta<1}$ is a strong reduction.

(ii) $(t_\gamma)_{\gamma<\beta+1}$ is a strong reduction if $(t_\gamma)_{\gamma<\beta}$ is a strong reduction.

(iii) $(t_\gamma)_{\gamma<\lambda}$ is a strong reduction if for all $\beta<\lambda$ the reduction $(t_\gamma)_{\gamma<\beta}$ is strong and $\forall d>0\,\exists\beta<\lambda\,\forall\gamma\,[\beta\leq\gamma<\lambda \Rightarrow d_\gamma>d]$.

A *strongly converging reduction* is a converging sequence that is a strong reduction. The *stable part* of an element t_β of a strongly converging reduction $(t_\beta)_{\beta\leq\omega}$ are the nodes at a depth less or equal to d_β.

NOTE. From now on, we will assume all reduction sequences to be strongly convergent ones. We will write $\to^\alpha$ for a strongly converging reduction of ordinal length α. Furthermore, we will write $\to^\omega$ for a strongly converging reduction sequence of length ω, $\to^{<\omega}$ for finite reduction sequences, and $\to^{\leq\omega}$ for strongly converging reduction sequences of length $\leq\omega$. We will write $\to^{\leq\omega}_n$ if a finite initial segment of the reduction sequence contains a needed reduction.

We do not need all the theory of transfinite reductions. An important result we will need, is the compressing lemma. It states that any strongly converging reduction sequence can be compressed in a strongly converging reduction sequence of length $\leq\omega$. Also the transfinite parallel moves lemma will be needed.

7.5. LEMMA (Compressing lemma). *Let $t \to^\alpha t'$ be a reduction sequence of length α. Then there is also a reduction sequence of length $\leq\omega$ such that $t \to^{\leq\omega} t'$.*

PROOF. See lemma 3.2.5. in Kennaway *et al.* [1990]. □

7.6. LEMMA (Transfinite parallel moves lemma). *Let $t \to^\alpha t_\alpha$ be a strongly converging reduction sequence, and let $t \to s_0$ be a reduction of a redex Δ in t. Then, for any $\beta\leq\alpha$, a term s_β can be constructed by outermost contraction of all descendants of Δ in t_β such that $s_\beta \twoheadrightarrow s_{\beta+1}$ and $t_\beta \to^{\leq\omega} s_\beta$. Also, the reductions in s_β form a strongly convergent reduction sequence to s_α and $t_\alpha \to^{\leq\omega} s_\alpha$.*

PROOF. See lemma 3.1.2. in Kennaway *et al.* [1990]. □

7.2. Transfinite reductions and constructor head normal forms

We will use $\rightarrow_n^{\leq\omega}$ (including $\bot$-reductions) as the new basis for approximations. First, however, we have to prove the necessary properties for this new reduction relation. In this section we will show that it is confluent and hypernormalising. Many proofs are similar to the proofs used in the standard theory of transfinite reductions. Therefore, we will be rather informal.

The notions of hnf and chnf extend to infinite terms in a rather straightforward way (that is, a term has a (c)hnf if there is a *finite* reduction sequence leading to a (c)hnf). We have the following important lemma, that states that it is always possible (if a term has a chnf) to get a chnf in a finite number of reduction steps.

7.7. LEMMA. *Let $t \rightarrow^{\leq\omega} t'$ such that t' is in constructor head normal form. Then there is a finite reduction sequence $t \rightarrow^{<\omega} s$ such that s is in chnf.*

PROOF. Because of strong convergence there has to be a finite (initial) reduction sequence to a term s such that $t \twoheadrightarrow s \rightarrow^{\leq\omega} t'$ and $root(s)$ is a constructor. □

The next important step is to prove that the extension of $\rightarrow_{\bot r}$ to transfinite reductions is confluent. As usual, we take $\rightarrow = \rightarrow_{\bot r}$. Inspection of the proofs for the compressing lemma and the transfinite parallel moves lemma shows that these lemmas can be extended to $\rightarrow$. We can now show confluence by showing that $\bot$-normal forms are unique.

7.8. DEFINITION ($\bot$-normal forms). A term is in $\bot$-normal form if it has no redexes other then $\bot$ with respect to $\rightarrow_{\bot r}$.

7.9. PROPOSITION. (i) *Any term in $T_\bot^\infty$ has a $\bot$-normal form.*

(ii) *$\rightarrow_{\bot r}^\alpha$ is confluent.*

PROOF. (i) Let $t \in T_\bot^\infty$. We will construct a strongly converging reduction to $\bot$-normal form. We first note that for any term there is a finite reduction sequence to a chnf, or to $\bot$ if it has no chnf. Now, let $t \rightarrow^{<\omega} c_0$ be such a reduction sequence. We can repeat this for all of the arguments of c_0. In this way, we construct a strongly converging reduction. That is, there are terms c_β such that $c_\beta \twoheadrightarrow c_{\beta+1}$ such that for c_β all nodes on a depth $\leq d_\beta$ are in chnf. The limit of this sequence is in $\bot$-normal form.

(ii) The proof is similar to the proof of lemma 4.1.2 in Kennaway *et al.* [1990]. Let t_c be the $\bot$-normal form of a term t. Any strongly convergent reduction sequence starting at t can be extended to a reduction to t_c. This can be proved with the help of the transfinite parallel moves lemma and the compressing lemma. □

We note that $\rightarrow_r^\alpha$ is not confluent, as is shown in Kennaway *et al.* [1990]. However, the causes for this non-confluence are exactly some reduction sequences that can never lead to a (head) normal form. These cases are handled by $\bot$-reductions.

We also have the notion of neededness for infinite terms. As a consequence of lemma 7.7 all reductions in a finite reduction sequence to a constructor head normal form occur within a certain finite depth of the intermediate terms. Now take a term $t \in T_\bot^\infty$ such that it has a chnf. Clearly, there is a finite term $s \leq t$ such that s has a chnf. As for the finite case, reductions in t can be simulated in s and vice versa. Now it is easy to see that a needed subterm in t corresponds to a needed subterm in s and vice versa (take a needed subterm in s. If the corresponding subterm in t is not needed, there would be a reduction sequence to chnf that doesn't reduce the needed subterm of s). A direct consequence is that theorem 4.3 extends to infinite terms. Furthermore, all terms having a chnf have a finite number of needed subterms.

The last we have to show is that $\rightarrow_n^{\leq\omega}$ is hypernormalising. Consider first the reduction sequence $t \rightarrow^\omega \cdot \rightarrow s$ such that the last reduction step is a needed one. It is easy to show that then $t \rightarrow_n^{\leq\omega} s$, that is, the needed reduction can be moved to an initial finite segment of the reduction sequence. The same holds, by applications of the compressing lemma and the above observation, for a reduction sequence $t \rightarrow_n^{\leq\omega} \cdot \rightarrow_n^{\leq\omega} s$: all needed reductions can be moved to the left. It is now obvious to show that $\rightarrow_n^{\leq\omega}$ is hypernormalising.

7.3. μ-Approximations

7.10. DEFINITION. Let $\mathcal{A}$ be an approximation. We can extend the approximation further to $\mathcal{A}_\mu$ as follows. An approximation $\mathcal{A}_\mu$ contains:

(i) A set of terms T_μ. Terms for this set are built as follows. Build terms as for T_α (see definition 2.2 and 5.1), but add the rule:

if $t_\alpha \in T_\mu$ such that this term contains exactly $k\,(\geq 0)$ free occurrences of the free variable x, thus $t_\alpha = C[x^k]$ then $\mu x.C[x^k] \in T_\mu$. Within the new term, called a *cyclic* term, x is bound.

The definition of occurrences is extended with an extra rule. Let u be an occurrence, then: $\mu x.C[x^k]/u = C[(\mu x.C[x^k])^k]/u$

(ii) An ordering relation $\leq_\alpha$. It is defined in two steps. First, define $\leq$ such that $t \leq t'$ is defined only if t contains no μ-terms. $\leq$ equals the previous $\leq_\alpha$ of definition 5.1 plus one new rule:

$$t \quad \leq \quad \mu x.C[x^k] \quad \text{if } t \leq C[(\mu x.C[x^k])^k]$$

Now we define $\leq_\alpha$ as:

$$t \leq_\alpha t' \quad \text{if for all } s \leq t \text{ it holds that } s \leq t'.$$

(iii) A reduction relation $\rightarrow_\alpha$. For $\rightarrow_\alpha$ it should hold that it simulates at least needed reductions, but takes account of converging infinite reductions. That is, let $t_\alpha \rightarrow_\alpha t'_\alpha$ and $t \in T_\perp$ such that $t \leq t_\alpha$. Then there is a $t' \in T_\perp^\infty$ such that $t' \leq t'_\alpha$ and $t \rightarrow_n^{\leq\omega} t'$.

Cyclic terms are somewhat peculiar. Let $t = \mu x.C[x^k]$. Then, we have $t \equiv \mu x.C[t^k]$. As follows from the definition of occurrences, a cyclic term has infinitely many valid occurrences. However, it is easy to see that a cyclic term only has a finite number of distinct subterms (two subterms might be syntactical unequal, but can be equivalent). This means that occurrence and subterm are no longer similar notions. However, this notion of occurrences will appear to be useful for embeddings.

NOTE.

(i) If Σ has a finite number of symbols, say $\Sigma = \{F_1, \ldots, F_n\}$, then $\top$ is equivalent to $\mu x.\langle F_1(x, \ldots, x), \ldots, F_n(x, \ldots, x)\rangle$. However, we prefer the use of $\top$.

(ii) There is a special term in T_μ: $\mu x.x$. It is equivalent to $\perp$.

(iii) The above notion of μ-terms is used more frequently for defining recursive structures. Usually, it is combined with a reduction relation $\mu x.C[x] \rightarrow C[\mu x.C[x]]$. However, we don't have necessarily $\mu x.C[x^k] \rightarrow_\alpha C[(\mu x.C[x^k])^k]$. Instead, the recursive behaviour of μ is obtained via the $\leq_\alpha$ operator.

(iv) The above definition of $\leq_\alpha$ seems not to be computable. However, it is possible to give a terminating algorithm (Nöcker [1993]).

7.11. LEMMA. (i) *$\leq_\alpha$ is a preordering (and a partial ordering if extended to equivalence classes).*

(ii) *Let $t_0 = \perp$, and $t_i = C[t_{i-1}^k]$. Then $t_i \leq t_l$, where $t_l = \mu x.C[x^k]$.*

PROOF. Straightforward. □

Most properties described in section 4 also hold for extended approximations. We will only consider some important differences:

(i) We add to lemma 5.2 the equivalence $\mu x.\langle x, t_1, \ldots, t_n\rangle \equiv \mu x.\langle t_1, \ldots, t_n\rangle$. The proof is easy.

(ii) The proof of transitivity for $\rightarrow_\alpha$ needs the compressing lemma: let $t_\alpha \rightarrow_\alpha t'_\alpha \rightarrow_\alpha t''_\alpha$. Let $t \in \gamma[t_\alpha]$. There is a $t' \in \gamma[t'_\alpha]$ such that $t \rightarrow_n s \rightarrow^{\leq\omega} t'$ for some term s. Similarly, there is a $t'' \in \gamma[t''_\alpha]$ such that $t' \rightarrow_n^{\leq\omega} t''$. By lemma 7.5 we have $t \rightarrow_n s \rightarrow^{\leq\omega} t''$, and thus $t \rightarrow_n^{\leq\omega} t''$.

(iii) For the construction of embeddings we have to add the rule:

if $t_\alpha = \mu x.C[x^k]$. Define for all $u \in O(t)$: $\varphi(u) = \varphi'(u)$, where $\varphi' : O(t) \to O(C[(\mu x.C[x^k])^k])$ an embedding (if $t_\alpha = \mu x.x$, take $\bot$ instead).

(iv) Bottom introduction is equivalent. The proof is equivalent to the proof of proposition 5.10. We note that it is important that $\to_n^{\leq\omega}$ is hypernormalising. We only have to take account of the case that $t \to_n^\omega t'$. However, then there is a term t'' such that $t \to_n t'' \to^\omega t'$ and such that after t'' no needed reductions are done. Now let $s = \mathcal{N}(t'')$ such that all needed redexes are represented in s. It is easy to see that $s \leq t'_\alpha$ and s, and thus t'', has a needed subterm to continue with.

We can now prove cycle-introduction. It has the same requirement, except for the neededness. The proof is now based on the construction of strongly convergent reduction sequences. First, however, we need a little lemma.

7.12. LEMMA. *Let $t \to_\alpha \langle t', t''\rangle$. If $t' \leq t$ then $t \to_\alpha t''$.*

PROOF. Choose a $s \in \gamma[t]$. We have to prove that there is a $s' \in \gamma[t'']$ such that $s \to_n^{\leq\omega} s'$. This can be shown be construction of a reduction sequence. There is a s_1 such that $s \to_n^{\leq\omega} s_1$ and either $s_1 \in \gamma[t']$ or $s_1 \in \gamma[t'']$. In the latter case we are ready. In the other case, $s_1 \in \gamma[t]$, and thus there is a next term s_2 such that $s_1 \to_n^{\leq\omega} s_2$ and $s_2 \in \gamma[\langle t', t''\rangle]$. We can construct a sequence of reductions $s_i \to_n^{\leq\omega} s_{i+1}$. Then there is either a $s_j \in \gamma[t'']$ (such that $s \to_n^{\leq\omega} s_j$), or the reduction sequence extends to infinity. Then, however, $s \to \bot$. □

7.13. PROPOSITION (Cycle introduction). *Let $t_\alpha \to_\alpha t'_\alpha$ such that t'_α has a non-empty set of disjoint subterms $\{s_1 \ldots s_k\}$ such that $s_i \leq t_\alpha$ for $1 \leq i \leq k$. Let $t'_\alpha = C_\alpha[s_1, \ldots, s_k]$. Then $t_\alpha \to_\alpha \mu x.C_\alpha[x^k]$ (where x is a fresh variable).*

PROOF. Let $t \in \gamma[t_\alpha]$. Then there is a $t' \in \gamma[t'_\alpha]$ such that $t \to_n^{\leq\omega} t'$. Let φ be an embedding $O(t') \to O(t'_\alpha)$. Now there are two possibilities:

(i) t' contains no subterm $s = t'/u$ such that $\varphi(u) = u_i$ (where u_i is the occurrence of s_i). Then the s_i can safely be replaced by any term, that is $t' \leq C_\alpha[\bot^k]$ (lemma 5.7.iii).

(ii) let $t' = t_1$. t_1 contains a subterm $r_1 = t_1/u$, $t_1 = C_1[r_1]$, such that $\varphi(u) = u_i$ and $C_1 \leq C_\alpha$. Then $r_1 \leq s_i \leq t_\alpha$. Then, either $r_1 = \bot$, or r_1 is a term not in chnf. Thus there is a reduction $r_1 \to_n^{\leq\omega} t_2$. Because of the compressing lemma, there is also a reduction $t_1 \to_n^{\leq\omega} C_1[t_2]$. We can repeat the argument for t_2.

We thus can make a sequence of terms and reduction steps as follows: $t_i = C_i[r_i]$, where $r_i \leq s_{i_j} \leq t_\alpha$, $r_i \to_n^{\leq\omega} t_{i+1}$ and $t_i \leq t'_\alpha$. Thus $t \to_n^{\leq\omega} C_1[C_2[\ldots C_i[r_i]\ldots]]$. This process stops if some of the $r_i = \bot$, or if it ends up in case (i). Consider first a finite sequence of m steps, thus $t \to_n^{\leq\omega} t_m = C_1[C_2[\ldots C_m[r_m]\ldots]]$. Since $C_i[a] \leq C_\alpha[a^k]$ for any term a, and since $C_m[r_m] \leq C_\alpha[\bot^k]$, we derive that $t_m \leq C_\alpha[(C_\alpha[\ldots C_\alpha[\bot^k]\ldots])^k] \leq \mu x.C_\alpha[x^k]$. If the construction leads to infinity, we have constructed a strongly convergent reduction sequence, of which the limit is contained in $\mu x.C_\alpha[x^k]$ We note that by application of lemma 7.12 we can choose an abstract term such that none of the C_i is empty (i.e. we can remove those s_i that are connected to the root via a path of unions). □

7.4. Extension of τ-reduction

τ-reduction remains the same as in section 6, except for cyclic terms. In the proof of lemma 6.2 we have to add the rule:

if $t = \mu x.C[x^k]$ Do $Match(p, C[(\mu x.C[x^k])^k)])$

(but we also have to relax the requirement for substitutions with respect to cyclic terms). It is important that a cyclic term has a finite number of subterms (though it can be considered as having an infinite structure). Then, also a cyclic term has a finite number of τ-redexes. Depending on the representation, several kinds of τ-reductions are possible. Clearly, the most compact representation of the cycle gives the most optimal reduction. This has as a consequence that one τ-reduction can simulate infinitely many standard reductions (in one concrete infinite term). For example:

Rewrite rules:

$$\begin{array}{lcl} \mathsf{F(x)} & \to & \mathsf{A(x)} \\ \mathsf{G(x)} & \to & \mathsf{B} \end{array}$$

Reductions:

$$\begin{array}{lcl} \mu\mathsf{x.}\langle \mathsf{F(x)},\ \mathsf{G(x)}\rangle & \to_\tau & \mu\mathsf{x.}\langle \mathsf{A(x)},\ \mathsf{G(x)}\rangle \\ \mathsf{F}(\mu\mathsf{x.F(x)}) & \to_\tau & \mathsf{F}(\mu\mathsf{x.A(x)}) \\ \mathsf{F}(\mu\mathsf{x.F(x)}) & \to_\tau & \mathsf{A}(\mu\mathsf{x.F(x)}) \end{array}$$

The last two reductions start with equivalent terms (both being equivalent to μx.F(x)). For obtaining a legal α-reduction it is sufficient to take the union of the results. However, this result is clearly worse than the reduct of μx.F(x) itself. Clearly, the abstract reduction mechanism is very sensitive for the choice of representatives of an abstract term.

Cyclic terms and cycle-introduction can be used in various ways. An important application is that we can use cyclic terms to represent possibly infinite concrete terms. For example, the term μx.Cons ($\top$,x) can be used to represent all infinite lists. Such a term can also be created during abstract reductions by cycle-introduction (i.e. if a function creates an infinite list). Consider for example the following function:

$$\begin{array}{lcl} \mathsf{Length\ (Cons\ (x,y))} & \to & +\ (1,\mathsf{Length\ (y)}) \\ \mathsf{Length\ (Nil)} & \to & 0 \end{array}$$

It is easy to show that Length is strict in its argument. However, it is also spine strict (the spine of the list can be reduced in advance). This can be derived by analysing reductions of Length (μx.Cons ($\top$,x)):

$$\begin{array}{lcll} \mathsf{Length}\ (\mu\mathsf{x.Cons}\ (\top,\mathsf{x})) & \to_\alpha & +\ (1,\mathsf{Length}\ (\mu\mathsf{x.Cons}\ (\top,\mathsf{x}))) & (\tau\text{-reduction}) \\ & \to_\alpha & \bot & (\bot\text{-introduction}) \end{array}$$

8. Discussion

We have developed a new framework for analysing strictness in orthogonal constructor term rewriting systems. We have shown correctness of some important properties of abstract reduction and of some analysis techniques. In section 6 we discussed a basis for a possible implementation. A more detailed treatment of implementation aspects is beyond the scope of this paper; the usefulness of abstract reduction is extensively discussed in Nöcker [1993].

One might ask in which sense it differs from abstract interpretation frameworks. Also, it is interesting in what way the work can be extended to our original goal: strictness analysis in functional languages (and term graph rewrite systems).

8.1. Strictness analysis by abstract interpretation

Most proposals for strictness analysis methods are based on some form of abstract interpretation, and are very different from our method (although we should mention the work of Sekar *et al.* [1990], which also uses an operational approach, based on term rewrite systems. However, they concentrate mainly on normal form strictness).

Many theoretical results are achieved in the field of abstract interpretation (e.g. Mycroft [1981], Burn *et al.* [1985], Wadler & Hughes [1987]). The main drawback of such approaches is that the fixed point determinations may consume very much computation time. One of the most important differences between abstract reduction and abstract interpretation is the moment on which approximations are applied. In abstract interpretation approximations are necessary when functions are converted to their abstractions. In abstract reductions approximations might be necessary during the reduction process. As a consequence, it is easy to give examples where an abstract reduction strictness analyser gives better results than an abstract interpretation strictness analyser. The main reason is that is always possible (given an abstract interpretation) to force approximations that are not necessary for abstract reduction. For example:

$$\begin{array}{lcl} \mathsf{Head\ (Cons\ (a,b))} & \to & \mathsf{a} \\ \mathsf{F\ (x)} & \to & \mathsf{Head\ (Cons\ (x,Nil))} \end{array}$$

With abstract reduction, it is straightforward to show that F is strict in its argument (by reducing F ($\bot$)). Within the abstract interpretation context it is not usual to include an abstract value for Cons ($\bot$,Nil), so that this value has to be approximated. On the other hand, it is also possible to give examples where abstract reduction finds less information.

For some part, the theory in this paper can be defined in terms of abstract interpretation. We can consider $T_\perp$ as concrete domain, and $T_\alpha^{\equiv}$ (or its μ-variant) as its abstraction. The two domains can be connected via a Galois connection (α, γ) defined as follows:

$$\begin{array}{lllll} \alpha{:}\mathcal{P}T_\perp \to T_\alpha^{\equiv} & \text{where} & \alpha(S) & = & \sqcup S \\ \gamma{:}T_\alpha^{\equiv} \to \mathcal{P}T_\perp & \text{where} & \gamma(t) & = & \{s \in T_\perp \mid s \leq t\} \end{array}$$

where $\sqcup$ is the least upper bound operator. However, as we already mentioned, $T_\alpha^{\equiv}$ is not a complete lattice, and the least upper bound doesn't need to exist always. There are several ways to complete the lattice (Goubault & Hankin [1991]), but they decrease the power of the analysis.

However, it seems difficult to adopt the reduction mechanism as we have described. In standard abstract interpretations, concrete functions are converted to abstract ones. Of the abstract functions fixed points are determined. Both the conversion as the fixed point analysis are not possible for our technique. Of course, there are alternatives. The technique developed in Hankin [1991] for the abstract interpretation of term graph rewrite systems is based on an approximation of a collecting semantics. For concrete applications it seems to complicated. A related technique has been used in Eekelen *et al.* [1991]. It turned out that the obtained results are rather limited.

8.2. Term graph rewriting

In this paper, we have restricted ourselves to the world of terms. Originally, abstract reduction was described in the context of term graph rewriting, or more specifically, in the context of the language Concurrent Clean. If we consider Clean as a functional language (which to some extent is possible), we can describe the semantics by using a term rewriting model. Why then, are we interested in a term graph rewriting model? There are three reasons. First, in an implementation of abstract reduction, sharing can be used as an optimisation of term rewriting. This is similar to using graph rewriting as an implementation for term rewriting. Lemma 5.4.iii indicates that it is indeed possible to share computations. Of course, it is possible to represent the μ-terms as cycles.

The second point is that we can add some new properties by using sharing. Consider for example the rule:

$$\mathsf{F}\ (1,2) \to \ldots$$

Now consider the term $\mathsf{F}\ (x{:}\top,x)$. That is, the arguments of F are shared. We can now consider this term either as representing all terms of the form $\mathsf{F}\ (x,y)$, or as representing all terms of the form $\mathsf{F}\ (x,x)$. In the second case the abstract term will not match the left-hand side of the rewrite rule. In other words, it is possible to keep track of all descendants of a certain term. This seems a nice optimisation, but it raises also some problems. For example, the term $\mu x.\mathsf{Cons}\ (\top,x)$ no longer needs to represent all infinite lists (since all list elements are shared).

Lastly, it seems very well possible to develop a similar theory for term graph rewriting itself. Of course, the problems introduced by sharing and cycles have to be studied carefully. Also, the notion of transfinite reductions needs some reconsideration. In special the relation between infinite terms and cyclic graphs is of interest. This is certainly a topic for further research.

8.3. Functional languages

Most functional languages are similar to constructor systems, i.e. there is a clear distinction between constructor and function symbols. This is one of the aspects of the type system. Typing is used to reject all 'illegal' terms. This means, that, if we forget about curried applications for the moment, most head normal forms will be constructor head normal forms. If this is not the case (e.g. selecting the head from an empty list), a run-time error is generated. For such cases, the head normal form property corresponds with the constructor head normal form property.

Currying is easily expressible in orthogonal term rewriting systems by adding explicit rules for the 'apply' function. For example, let F be a function with arity 2. We add the following rules to the rewrite system (where F_0 is the constructor to be used in a curried case):

$$\mathsf{Ap}\ (\mathsf{Ap}\ (\mathsf{F}_0,x),\ y) \quad \to \quad \mathsf{F}\ (x,y)$$

This function cannot be part of a function-constructor system (but yet of an orthogonal term rewriting system). Also, it is undecidable whether an application of apply is in head normal form. So, currying seems to be a big problem for abstract reduction. However, in the case that the apply function is the only exception in the function-constructor system, we can make a decidable approximation of the head normal form property, a kind of weak head normal form (whnf) property (an apply term is in whnf if its first argument is in whnf or in chnf). Properties as derived for the chnf case, can also be derived for the whnf case. In general, we believe that abstract reduction is possible for any orthogonal term rewriting system for which a decidable (and useful) approximation of head normal forms can be given. Clearly, typing is a possible method.

With respect to typing, we finally note that there are various ways how typing information can be used in an implementation. The analysis for objects of a specific type (e.g. lists, basic values, tuples) can be adapted in order to get a better run-time behaviour, more information etcetera.

Another important difference between functional languages and orthogonal term rewriting systems is the reduction strategy. The strategy determines what redex in a term is the next one going to be rewritten. In practice, a strategy determines evaluations of the arguments of a function application, if that application itself has to be reduced. A well-known strategy is the so-called functional strategy, used in languages like Miranda (Turner [1985]) and Concurrent Clean (Nöcker *et al.* [1991]). It reduces terms to head normal form before pattern matching, whereas patterns in a rewrite rule are tried from top to bottom, and from left to right. In orthogonal term rewriting systems, this strategy certainly doesn't need to be normalising. On the other hand, many function definitions in a functional program are not purely orthogonal. In fact, one of the effects of the strategy is to ensure a kind of orthogonality (or, in other words, confluence of the reduction).

In our framework there are several ways to take account of the reduction strategy. It would be nice if the starting point, orthogonal term rewriting systems, could be replaced by rewrite systems with a reduction strategy such that confluence is ensured. We believe that similar results can be obtained (but note that still some kind of head normal forms is needed). This is also a topic of further research. From an implementation viewpoint, a reduction strategy can be used to optimise abstract reduction. How this can be done for the functional strategy is described in Nöcker [1993].

9. References

Abramsky S., D. Gabbay and T. Maibaum, *Handbook of Logic in Computer Science*, volume I, Oxford University Press, 1992.

Arnold A. and M. Nivat, The Metric Space of Infinite Trees, Algebraic and Topological Properties, *Fundamenta Informatica*, 4, 1980.

Barendregt H.P., *The Lambda Calculus: its syntax and semantics*, volume 103 of *Studies in Logic*, North-Holland, Amsterdam, 1984.

Barendregt H.P., M.C.J.D. van Eekelen, J.R.W. Glauert, J.R. Kennaway, M.J. Plasmeijer and M.R. Sleep, Term Graph Reduction, in: *Proc. of Parallel Architectures and Languages Europe (PARLE)*, pages 141–158, Eindhoven, The Netherlands, Springer Verlag, LNCS 259 II, 1987.

Burn G.L., C.L. Hankin and S. Abramsky, The Theory of Strictness Analysis for Higher Order Functions, in: *Proc. of Workshop on Programs as Data Objects*, pages 42–62, DIKU, Denmark, Springer Verlag, LNCS 217, 1985.

Eekelen M.C.J.D. van, E. Goubault, C. Hankin and E.G.J.M.H. Nöcker, Abstract Reduction: Towards a Theory via Abstract Interpretation, in: Sleep Plasmeijer and Eekelen, editors, *Proc. of Semagraph Symposium*, Nijmegen, The Netherlands, 1991.

Goubault E. and C. Hankin, A Lattice for the Abstract Interpretation of Term Graph Rewrite Systems, in: *Proc. of Semagraph Symposium*, pages 263–273, Nijmegen, The Netherlands, 1991.

Hankin C., Static Analysis of Term Graph Rewriting, in: Aarts Leeuwen and Rem, editors, *Proc. of Parallel Architectures and Languages Europe (PARLE'91)*, pages 367–385, Eindhoven, The Netherlands, Springer Verlag, LNCS 506, 1991.

Huet G. and J.J. Lévy, Call by Need Computations in Non-Ambiguous Linear Term Rewriting Systems, Technical Report 359, INRIA, 1979.

Kennaway J.R. and M.R. Sleep, Neededness is Hypernormalising in Regular Combinatory Reduction Systems, Technical Report Preprint, School of Information Systems, University of East Anglia, Norwich, 1989.

Kennaway J.R., J.W. Klop, M.R. Sleep and F.J. de Vries, Transfinite Reductions in Orthogonal Term Rewriting Systems, Technical Report CS9041, CWI, 1990.

Klop J.W., Term Rewriting Systems, in: *in Abramsky* et al. *[1992]*, 1992.

Mycroft A., *Abstract interpretation and optimising transformations for applicative programs*, PhD thesis, University of Edinburgh, 1981.

Nöcker E.G.J.M.H., Strictness Analysis based on Abstract Reduction, in: *Proc. of Second International Workshop on Implementation of Functional Languages on Parallel Architectures*, pages 297–321, University of Nijmegen, the Netherlands, Technical Report 90-16, 1990.

Nöcker E.G.J.M.H., Strictness Analysis using Abstract Reduction, in: *Proc. of Conference on Functional Programming Languages and Computer Architecture (FPCA '93)*, Kopenhagen, ACM Press,, 1993.

Nöcker E.G.J.M.H., J.E.W. Smetsers, M.C.J.D. van Eekelen and M.J. Plasmeijer, Concurrent Clean, in: *Proc. of Parallel Architectures and Languages Europe (PARLE'91)*, pages 202–219, Eindhoven, The Netherlands, Springer Verlag, LNCS 505, 1991.

Sekar R.C., Shaunak Pawagi and I.V. Ramarkrishnan, Small Domains Spell Fast Strictness Analysis, in: *Proc. of Symposium on Principles of Programming Languages (POPL)*, pages 169–183, San Francisco, USA, ACM Press,, 1990.

Sleep M.R., M.J. Plasmeijer and M.C.J.D. van Eekelen, *Term Graph Rewriting, Theory and Practice*, John Wiley and Sons inc., 1993.

Turner D.A., Miranda: A non-strict functional language with polymorphic types, in: *Proc. of Conference on Functional Programming Languages and Computer Architecture (FPCA '85)*, pages 1–16, Nancy, France, Springer Verlag, LNCS 201, 1985.

Wadler P. and R. J. M. Hughes, Projections for strictness analysis, in: *Proc. of Conference on Functional Programming Languages and Computer Architecture (FPCA '87)*, pages 385–407, Portland, Oregon, USA, Springer Verlag, LNCS 274, 1987.