Analysing resource use in the λ-calculus by type inference

*S. A. Courtenage** *and C. D. Clack*
{scourten,clack}@cs.ucl.ac.uk
Dept of Computer Science, University College London,
London, WC1E 6BT, ENGLAND

Abstract

If we view functions as processes, then their resources are their arguments, supplied through application, and used by the function to produce a result. In this paper, we define resource use for functions, based on the syntactic notion of needed redexes from [BKKS86]. We introduce a variant of neededness, *tail-neededness*, and define packets of needed descendants of redexes in order to measure the degree of neededness. These results are generalised to produce a semantic characterisation of the resource use properties of functions, using a term-model. By means of the Curry-Howard isomorphism, we apply these ideas to proof trees of propositions in Intuitionistic Logic to demonstrate that propositions, i.e. types, can be used to express the usage properties of functions. A *resource-aware* type system capable of inferring such types for λ-terms is presented.

1 Introduction

Despite the many advances made in the implementation of lazy functional languages, static program analysis remains an important topic as researchers seek to match the conceptual advantages in programming with adequate run-time performance. The goal of static analysis is to provide information about expected program behaviour which can be used within a compiler to translate the program into a more optimal form.

There are two principal methodologies and many methods of static analysis. The two methodologies are Abstract Interpretation [AH87] and Type Inference [Jen92] [Bur92]. Among the methods of analysis, Strictness Analysis [CPJ85] [BHA86] and variations on compile-time garbage collection [Blo89] [Hug91] [Arg91] have received most attention, for their advantages with regard to increased parallelism and store-use optimisation. (Chapter 4 of [Hug91] contains a useful taxonomy of the field.)

Analysis by type inference is split into two camps: type inference over abstract domains [Jen91] [KM89]; and *resource-aware* type systems [Wri92] [Bak92] [Bie92], inspired by Linear Logic [Gir87] [GSS92], with which our work is directly concerned. (There also exist type systems for languages based on Linear Logic; see [Lin92] [Wad91] [Mac91].)

The aim of our resource-aware type system is to uncover the structural rules of the system of logic underlying type inference which are responsible for sharing and discarding of resources. In the context of the λ-calculus, resources are function arguments, and in the context of types, they are hypotheses to typing judgements and, by implication, the source types of function types. Our type system has an important advantage over analyses based on Abstract Interpretation, and allied type inference systems. The abstract domains of such analyses are defined with reference to specific optimisation methods, so that, for example, different analyses must be made to obtain strictness and sharing information. Our type system, however, by focusing on the usage made of values from domains, rather than the structure of the domains themselves, provides information about program behaviour independent of the optimisation to be performed. The information provided by the type system presented in this paper could potentially subsume analyses for strictness, sharing and single-threading of data structures.

This paper's contribution is to define resource-use in the λ-calculus, based on a syntactic notion of needed redexes from [BKKS86], and to show, using a term-model, that our definition of usage properly characterises the resource use of the semantic functions denoted by λ-terms. Furthermore, we show that types can be used to express resource use properties of functions by demonstrating an equivalent definition of usage at the level of proofs of types. We conclude with a brief presentation of a resource-aware type inference system. The syntax of this system closely resembles that of [Wri92] [Bak92] [Bie92], but this paper offers key insights into its semantics.

2 Background

We recount some of the basic tenets of the λ-calculus necessary to an understanding of this paper.

The set Λ of λ-terms is defined inductively by

$$\begin{array}{l} \mathtt{x} \in \Lambda \\ \mathtt{M},\mathtt{N} \in \Lambda \Rightarrow (\mathtt{MN}) \in \Lambda \\ \mathtt{M} \in \Lambda \Rightarrow \lambda\mathtt{x}.\mathtt{M} \in \Lambda \end{array}$$

*During the period of this work, S. A. Courtenage was supported by a SERC studentship

i.e. λ-terms are formed from variables, applications of terms and abstractions created by binding a variable over a term.

A term $(\lambda\mathtt{x}.\mathtt{M})\mathtt{N}$ is a *redex*. The result of *contracting* such a redex is $\mathtt{M}[\mathtt{x} := \mathtt{N}]$, in which N is substituted for all free occurrences of x in M (technical details regarding renaming of variables are omitted here). A term is in *normal-form* (nf) if it contains no redexes. The leftmost redex in a term is a head redex if it is only preceded by λs and bound variables; for example, PR in $\lambda\mathtt{x}.(\mathtt{P}\ \mathtt{R})\ \mathtt{M}$. A term is in head-normal form (hnf) if the head subterm does not contain a redex; for example $\lambda\mathtt{x}.\mathtt{y}\ \mathtt{M}$ is in head-normal form. *Reduction* describes a sequence of one or more contractions and is represented by $\twoheadrightarrow_\beta$. Leftmost reduction describes a reduction sequence in which only leftmost redexes are contracted; head reduction is similarly defined. *Conversion* is an equivalence relation based on reduction, and is represented by $=_\beta$. We will denote the nf of a term M by $\mathtt{M}_{nf}$ (similarly for hnf).

Models are used to provide a semantics of the λ-calculus. In this paper, we will require only the very simplest model, the *term-model*. This is defined as a triple $< D, \cdot, [\![\ ]\!] >$, in which D is a non-empty set of convertibility classes of terms, formed by

$$D = \{[\mathtt{M}] | \mathtt{M} \in \Lambda\}$$
$$[\mathtt{M}] = \{\mathtt{N} | \mathtt{N} =_\beta \mathtt{M}\}$$

and $\cdot$ is a binary operator of type $D \times D \rightarrow D$, and $[\![\ ]\!]$ maps λ-terms to D in the context of an environment ρ assigning values of D to variables.

The term-model is essentially trivial, being a reflection simply of the syntax of λ-terms [HS86], but is useful to our purpose, as our definition of usage begins syntactically. A term-model is also useful in that it is the canonical model for proofs of soundness and completeness of type systems.

In the latter part of this paper, we will refer to the implicative fragment of Intuitionistic Logic (IL). This is essentially Curry's system of F-Deducibility [Hin83] with Structural rules for Exchange, Contraction and Weakening of hypotheses. The presentation will use sequents in the form

$$\Gamma \vdash \sigma$$

where Γ is understood to be an unordered set or *base* of hypotheses $\alpha, \beta, \ldots$; $\vdash$ is meant to represent deduction; and σ is the proposition deduced from the base of hypotheses. IL is presented in Appendix A.

3 Neededness and usage in the λ-calculus

In this section, we define a semantics of resource use for functions represented by terms in the λ-calculus. Our aim is a model of the λ-calculus which will enable us to analyse the use that functions make of their resources, i.e. their arguments.

We begin with a syntactic description of use in terms of needed redexes in λ-terms. Building on the work of [BKKS86], we define a new form of neededness, *tail-neededness*, and introduce *packets* of needed descendants of a redex in order to measure the degree to which that redex is needed. We show that this syntactic description of neededness can be the basis of a semantic definition by construction of a term model.

Following on from this, we define a small, abstract domain of uses based on ranges of sizes of packets of needed descendants of redexes, which can be used to characterise functions according to their argument use.

3.1 Neededness in the λ-calculus

Neededness and head-neededness are defined using a notion of *descendant* to determine which redexes within a term are contracted during the reduction of the term.

Definition 3.1 (Definition 2.4 of [BKKS86]) *The descendants of a sub-term* N *in* M*, after the reduction* $\mathtt{M} \rightarrow \mathtt{M}'$*, are those sub-terms of* $\mathtt{M}'$ *that can be* traced back *to* N*. If* $\mathtt{N} \equiv \mathtt{x}$*, then the descendants of* N *in* $\mathtt{M}'$ *are occurrences of* x *in* $\mathtt{M}'$*. If* N $\equiv (\lambda\mathtt{x}.\mathtt{E})$ *or* $(\mathtt{E}_1\ \mathtt{E}_2)$*, then the descendants of* N *in* $\mathtt{M}'$ *are those sub-terms with the same outermost pair of brackets as* N*. (A more precise definition can be made using labels, for which see [Klo80] or [Bar84].)*

Definition 3.2 (Definition 3.1 of [BKKS86]) *If* R *is a redex in* M*, then*

- R *is* needed *in* M *if every reduction sequence of* M *to nf reduces a descendant of* R.
- R *is* head-needed *in* M *if every reduction sequence of* M *to hnf reduces a descendant of* R.

For example, in the term

$$\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{I}\ \mathtt{x}\ (\mathtt{K}\ \mathtt{y}\ (\mathtt{I}\ \mathtt{y}))$$

the redex I x is needed and head-needed, while the redex (K y (I y)) is needed but not head-needed. (This is Example 3.2 of [BKKS86].)

A corollary of the definitions of neededness and head-neededness is that every head-needed redex is also needed, since every reduction to nf contains a reduction to hnf [BKKS86]. What about those redexes that are needed but not head-needed? We will see later on that such redexes are important to our defining usage in the presence of higher-order functions, and hence we introduce a new definition to characterise them.

Definition 3.3 *If* R *is a redex in* M *then* R *is tail-needed in* M *iff* R *is needed but not head-needed in* M

Lemma 3.1 *Let* R *be a tail-needed redex in* M*. Then* R *occurs as an argument term, or as a sub-term of an argument term, to a higher-order, i.e. function-valued, variable in* M.

Proof 3.1 *By the definitions of head normal-form and tail-neededness,* R *is not in the head-position of* M.
□

We could adopt head-neededness as the basis of our definition of usage, in which case a function would be said to use its argument if all redexes substituted for the argument variable were head-needed. In essence, this is strictness analysis, as noted by Barendregt [BKKS86]. (This is in fact the semantic basis of the work by Wright in developing analysis of strictness by type inference [Wri92].)

However, a more interesting question is how often an argument is used; for example, is it used only once, more than once, or not at all. Answers to this question provide more detailed information about the computational behaviour of a function, consequently allowing us to perform more interesting optimisations in our implementation than simple strictness analysis.

If we ask how often a redex is needed in the reduction of a term to nf, then we are really asking how many of its descendants are needed. For example, in the term

$$(\lambda \mathtt{f}.\lambda \mathtt{x}.\mathtt{f}(\mathtt{f}\mathtt{x}))\mathtt{R}$$

where $\mathtt{R} \equiv \mathtt{SKK}$, two descendants of R will occur as needed redexes in the leftmost reduction to nf. To capture this idea of the number of times a redex may be needed, we introduce the notion of *packets.*

Definition 3.4 *Let* R *be a needed redex in* M. *Then the* packet *of* R *will be the set of descendants of* R *that are contracted in the* leftmost *reduction of* M *to* $\mathtt{M}_{nf}$. *The size of the packet of* R*, denoted by* $|\mathtt{R}|$*, is the number of descendants of* R *that occur as needed redexes.*

For the example above, using integers as labels for descendants, the reduction sequence to nf is

$$\begin{array}{ll} & (\lambda \mathtt{f}.\lambda \mathtt{x}.\mathtt{f}\ (\mathtt{f}\ \mathtt{x})\ (\mathtt{SKK}) \\ \Rightarrow & \lambda \mathtt{x}.(\mathtt{SKK})^1\ ((\mathtt{SKK})^2\ \mathtt{x}) \\ \Rightarrow & \lambda \mathtt{x}.\mathtt{I}\ ((\mathtt{SKK})^2\ \mathtt{x}) \\ \Rightarrow & \lambda \mathtt{x}.(\mathtt{SKK})^2\ \mathtt{x} \\ \Rightarrow & \lambda \mathtt{x}.\mathtt{I}\ \mathtt{x} \\ \Rightarrow & \lambda \mathtt{x}.\mathtt{x} \end{array}$$

In this case, the packet of $(\mathtt{SKK}) = \{(\mathtt{SKK})^1, (\mathtt{SKK})^2\}$ and $|(\mathtt{SKK})| = 2$.

It is possible that reduction orders other than leftmost reduction could contract fewer descendants of the needed redex. In the example above, an applicative reduction order, in which the redex (SKK) is contracted on the first step would yield a set of descendants contracted on the path to nf of size 1. In other words, packets do not necessarily provide a true picture of the degree of neededness for redexes over all possible reduction paths; in fact, the most we can say is that, by definition of neededness, if a packet is non-empty on the leftmost path then it will be non-empty on all possible reduction paths. Packets of needed descendants are a *reliable* assessment of the degree of neededness, however, in the sense that for all terms possessing a normal form, packets measure how often a term is needed along a reduction path guaranteed to terminate and find the normal form.

Definition 3.5 *Let* R *be a needed redex in* M. *The* head packet *of* R *is the set of descendants that are contracted in the* head *reduction of* M *to* $\mathtt{M}_{hnf}$. *The* tail packet *is the set of descendants contracted in the* leftmost *reduction of* $\mathtt{M}_{hnf}$ *to* $\mathtt{M}_{nf}$. *The sizes of the head and tail packets of* R *will be denoted by* $|\mathtt{R}|_h$ *and* $|\mathtt{R}|_t$*, respectively.*

Notation 3.1 *Say* $|\mathtt{R}| \in \mathtt{MR}$ *to denote the size of the packet of* R *in the reduction of* MR *to nf by leftmost reduction. Similarly for* $|\mathtt{R}|_h$ *and* $|\mathtt{R}|_t$.

So far, we have assumed that reduction sequences terminate, i.e. that terms have a normal form. But what if this is not the case? In so far as packet sizes are concerned, we note only that for non-terminating reduction sequences, packets are undefined, i.e. $|\mathtt{R}| = \perp$.

We finish this discussion of neededness and descendants by looking at two properties of packets that will be useful in our later discussion of usage. The first concerns the merger of packets of a redex in the case where descendants of the redex occur in the sub-terms of an application MN, and the second demonstrates the effect of the neededness of a redex on a redex sub-term.

Lemma 3.2 *Let* $\mathtt{M} \equiv \mathtt{PQ}$ *and let* R *be a redex in* Q. *Given* $|\mathtt{R}| \in \mathtt{Q}$ *and* $|\mathtt{Q}| \in \mathtt{M}$*, then*

$$|\mathtt{R}| \in \mathtt{M} = |\mathtt{R}| \in \mathtt{Q} * |\mathtt{Q}| \in \mathtt{M}$$

Proof 3.2 *Obvious*
□

Lemma 3.3 *Let* $\mathtt{M} \equiv \mathtt{PQ}$. *Let there be descendants of a redex* R *in* P *and in* Q. *Then*

$$|\mathtt{R}| \in \mathtt{M} = |\mathtt{R}| \in \mathtt{P} + (|\mathtt{R}| \in \mathtt{Q} * |\mathtt{Q}| \in \mathtt{M})$$

Proof 3.3 *By Definition 3.4 and Lemma 3.2*
□

3.2 Neededness and term models

As an extension to the work in Section 3.1, we now look at neededness and packets in a term model of the λ-calculus (see [Bar84] [HS86] for discussions of models of the λ-calculus).

Results from [BKKS86] show that (head-) neededness is a persistent property. In particular, if a redex R is needed in MR, then it is needed in $\mathtt{M}'\mathtt{R}$ for all $\mathtt{M} =_\beta \mathtt{M}'$.

Lemma 3.4 (Lemma 3.11 of [BKKS86]) *If* $\mathtt{F} =_\beta \mathtt{F}'$*, then* R *(head-)needed in* FR $\Rightarrow$ R *(head-)needed in* $\mathtt{F}'\mathtt{R}$.

Proof 3.4 *See* [BKKS86]
□

As a consequence, if a redex R is (head-)needed in XR, then

$$\forall \mathtt{Y} \in [\mathtt{X}] \Rightarrow \mathtt{R} \text{ is needed in } \mathtt{YR}$$

In other words, if R is (head-)needed in XR, then R is (head-)needed when applied to all the terms in the convertibility class of X. To represent this, we will write

$$\mathtt{R}\ \text{head-needed in}\ [\mathtt{X}]\mathtt{R}$$

where, by [X], we understand the canonical element of the convertibility class for X.

The preservation of neededness by β-conversion also extends to packets, as the following lemma and proof demonstrate.

Lemma 3.5 *If* $\mathtt{M} =_\beta \mathtt{N}$ *then, given a redex* R*,*

$$|\mathtt{R}| \in \mathtt{MR} = |\mathtt{R}| \in \mathtt{NR}$$

The proof of this theorem proceeds along lines similar to those for the proof of Lemma 3.4.

Proof 3.5 *As for Proof 3.4*
□

The persistence of head-neededness over reduction (Proposition 3.7 of [BKKS86]) mean that Theorem 3.5 also applies to head packets $|\mathtt{R}|_h$. Theorem 3.5 also applies, by extension, to tail packets $|\mathtt{R}|_t$.

We can infer a consequence of Lemma 3.5 similar to that which followed from Lemma 3.4, i.e. that given some redex R, if R is (head-)needed in XR, then

$$\forall \mathtt{Y} \in [\mathtt{X}] \Rightarrow |\mathtt{R}| \in \mathtt{YR} = |\mathtt{R}| \in \mathtt{XR}$$

By an abuse of notation, we will now refer to

$$|\mathtt{R}| \in [\mathtt{X}]\ \mathtt{R}$$

This notation will also allow us to refer to $|\mathtt{R}| \in d \cdot \mathtt{R}$ for $d \in D$ in the term-model. (Again, the same holds for head and tail packets.)

If [R] is the convertibility class for some redex R, can we infer, modulo conversion, that, given a term M,

$$\forall \mathtt{R}_i, \mathtt{R}_j \in [\mathtt{R}] \Rightarrow (|R_i| \in \mathtt{MR_i}) = (|R_j| \in \mathtt{MR_j})$$

i.e. that the same descendants of R, modulo conversion, are needed? The answer is yes, although we will have to modify slightly the definition of neededness since not all the terms in the convertibility class will be redexes. (For example, the normal form of R, $\mathtt{R}_{nf}$, is in [R], and, by definition of normal form, cannot be contracted and hence needed in the usual sense.)

Definition 3.6 *Let* R *be a sub-term of* M*. Then*

- R *is a needed sub-term of* M *iff a descendant of* R *appears as the leftmost sub-term in a redex contracted in the leftmost reduction of* M *to* $\mathtt{M}_{nf}$ *or a descendant of* R *appears in* $\mathtt{M}_{nf}$
- R *is a head-needed sub-term of* M *iff a descendant of* R *appears as the leftmost sub-term in a redex contracted in the head reduction of* M *to* $\mathtt{M}_{hnf}$ *or a descendant of* R *appears in the head-position of* $\mathtt{M}_{hnf}$

For example, in the term

$$\lambda \mathtt{f}.\lambda \mathtt{x}.(\mathtt{I}\ \mathtt{f})\ \mathtt{M}$$

I appears as a head-needed (and needed) subterm. In the term

$$\lambda \mathtt{x}.\mathtt{f}\ ((\lambda \mathtt{y}.\lambda \mathtt{z}.\mathtt{y})\ \mathtt{N}\ \mathtt{M})$$

ther sub-term λy.λz.y is needed but not head-needed.

Note that the leftmost subterm in a redex can be the redex itself in those cases where the subterm is not in normal-form.

An alternative definition can be made using the persistent labels of [Bak92], in which case neededness of sub-terms is indicated by the presence of labels after reduction to normal-form.

A definition of tail-needed sub-terms can be made relative to the definitions of needed and head-needed sub-terms, as for tail-needed redexes.

We can now consider convertibility classes of arguments R in MR, since the identification of the descendant is unaffected by any conversions of the term itself. In other words, we can now consider neededness and packets for applications

$$[\mathtt{M}] \cdot [\mathtt{R}]$$

We can therefore extend the notation introduced above to refer to

$$|e| \in d \cdot e$$

for d and $e \in D$ in the term-model $\mathcal{D}$. (Again, similarly for head and tail packets).

What interpretation can we place upon $|e| \in d \cdot e$? The value of $|e|$ is the number of times the argument e *may* be needed by the function d. The value of $|e|_h$ is the number of times that e is definitely needed by d, while $|e|_t$ reflects only the possibility of neededness engendered by the presence of higher-order functions.

3.3 Usage in the λ-calculus

In the previous section, we developed the means by which we could measure the number of times a λ-term was needed during reduction. Neededness, however, is a property of arguments to functions, rather than of the functions themselves, as shown by the fact that different arguments may have differently-sized head and tail packets of needed descendants even though the same function is applied to them. For example, λx.x and λx.y possess packets of different sizes when passed to the function λf.f (f x).

In this section, therefore, we develop a definition of resource-use based on the results of the previous section, but which describes the properties of functions independent of the values of particular arguments. As a result, we produce a small, abstract domain of use values which can be used to characterise argument usage of functions.

The mechanism for defining resource use based on neededness and packets is relatively simple. It is based on the observation made above that the number of times an argument is needed may

Usage	Head packet size	Tail packet size
0+	$0 - \infty$	$0 - \infty$
1+	$1 - \infty$	$0 - \infty$
1	1	0
0	0	0
$\perp$	$\perp$	$\perp$

Table 1: Usage values as ranges of head and tail packets

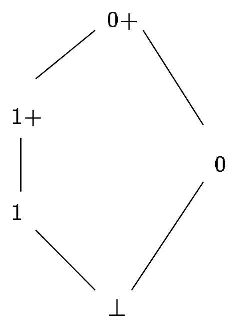

Figure 1: Domain $\mathcal{U}$ of usage annotations

depend upon its own value, and on Lemma 3.1. Briefly, both observation and lemma imply that precise information about argument use is in general unavailable, except in special cases. These special cases are when an argument does not depend upon its own value, for example the argument f in the Church numeral

$$\bar{1} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\texttt{f x}$$

and when the tail-needed packet for the argument is empty (again, as in the case of f in $\bar{1}$).

To represent resource use in the presence of such uncertainty then, we define usage in terms of *ranges* of possible sizes of head and tail packets of arguments. Table 1 defines use values using such ranges, represented as sets of packet sizes, and Figure 1 presents the use values as a domain ordered by subset inclusion over the ranges.

To say then that a function λx.M has, for example, usage 1+ means that any argument substituted for x on application will have head and tail packets of needed descendants in the range of sizes defined for 1+. An intuitive explanation of the points of the domain is:

- 0+ The argument is used 0, once or many times by the function
- 1+ The argument is used *at least once* by the function
- 1 The argument is used exactly once
- 0 The argument is *unused*
- $\perp$ Undefined

Both the operators + and $*$ defined for packets in Section 3.1 extend to usage values, by applying them to the ranges of head and tail packets underlying the definition of usage. For example:

$$\begin{array}{lcl} 1+1 & = & 1+ \\ 1+0 & = & 1 \\ (1+)*0 & = & 0 \end{array}$$

Note that + and $*$ over usage values are defined in such a way as to find the least point in the $\mathcal{U}$ domain of usage values that is described by the result of applying + and $*$ over the neededness values in the ranges underlying the definition of usage values.

4 Neededness and usage in proofs

Having defined a semantic basis for usage, it would be useful to be able to deduce the use properties of functions by means of some analysis. In this section, we show that type inference can be used as such an analysis, by demonstrating the facility and expressiveness of types to represent the use properties of functions.

Our approach shall be to consider the implicative fragment of propositional Intuitionistic Logic (IL), and develop equivalent notions of neededness and usage at the level of proofs. We shall then relate these results to types by means of the Curry-Howard Isomorphism [How80], which identifies propositions, proofs and elimination of redundant proof steps in IL with types,terms and reduction in the typed λ-calculus.

4.1 Descendants of proofs in proof normalisation

A proof P of a proposition $\Gamma \vdash \sigma$ may be represented as a tree of deductions [Gir89]. Any branch of a proof-tree representing P is itself a proof-tree, here called a sub-proof.

In a manner equivalent to the tracing of descendants in the λ-calculus, we define descendants of sub-proofs in IL.

Definition 4.1 *The descendants of a sub-proof* Q *whose conclusion is* $\Gamma \vdash \alpha$ *in a proof* P *whose conclusion is* $\Lambda \vdash \sigma$*, after the normalisation of the proof of* P *to* P'*, are all those sub-proofs of* P' *that can be traced back to* Q*, identified at the level of the rule used to deduce the conclusion to the sub-proof. (As for descendants of* λ*-terms in [Bar84] [Klo80], underlining or labelling may be used to make a more precise identification.)*

In the λ-calculus, the number of descendants of a term decreases or increases as it or an enclosing term are substituted for occurrences of bound variables in the contraction of higher-level redexes during reduction.

In proof-theoretic terms, descendants of sub-proofs involved in the elimination of a redundant proof step, and the consequent rewriting of the proof tree, are created or erased in an equivalent manner, according to the size of the parcel for the hypothesis [Gir89] that the sub-proof replaces during the normalisation step. The Axiom rule introduces a hypothesis with a parcel of size 1, while Contraction and Weakening respectively merge parcels into larger ones and introduce dummy or empty parcels [Gir89]. Note that sub-proofs are erased or discarded when substituted for a hypothesis introduced using Weakening, i.e. one represented by a dummy parcel.

4.2 Neededness in proofs

We restate the main definitions of neededness from [BKKS86] in proof-theoretic terms.

Definition 4.2 *If* P *is a sub-proof in a proof* Q*, then* P *is needed iff every proof normalisation of* Q *to normal-form involves a normalisation of a descendant of* P*.*

We can also define a proof-theoretic version of head neededness, by means of a simple transliteration of the definition for λ-terms.

Definition 4.3 *The head sub-proof of a proof* Q *is that part of the left-most branch of* Q *reached by first any continuous sequence of* $\to$ Intro *rules followed by any continuous sequence of* $\to$*-E rules from the root of* Q*.*

This definition simply reflects the definition of the head sub-term of a λ-term: i.e., the leftmost sub-term to the right of any abstractors λ and their bound variables.

Definition 4.4 *A proof* P *is in* head normal form *iff the head sub-proof of* P *is in normal-form.*

A head-normal form for a proof is therefore one whose left-most branch from the conclusion of the proof also contains no redundant proof steps.

For example, the following proof is in head-normal form

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(\alpha \to \beta) \vdash \alpha \to \beta} \qquad \dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha}}{\vdash \alpha \to \alpha} \to \text{Intro} \quad \dfrac{}{\alpha \vdash \alpha}}{\alpha \vdash \alpha} \to \text{Elim}}{(\alpha \to \beta), \alpha \vdash \beta} \to \text{Elim}}{(\alpha \to \beta) \vdash \alpha \to \beta} \to \text{Intro}}{\vdash (\alpha \to \beta) \to \alpha \to \beta} \to \text{Intro}$$

(the labels on the Axiom rules have been omitted for brevity of presentation). The following proof, however, is not in head-normal form, since it contains the redundant proof steps $\to$ Intro_1 and $\to$ Elim_1:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha \vdash \alpha}\text{Axiom}}{\vdash \alpha \to \alpha} \to \text{Intro}_1 \quad \dfrac{}{\alpha \vdash \alpha}\text{Axiom}}{\alpha \vdash \alpha}\text{Elim}_1}{\vdash \alpha \to \alpha} \to \text{Intro} \qquad \dfrac{}{\alpha \vdash \alpha}\text{Axiom}}{\alpha \vdash \alpha} \to \text{Elim}}{\vdash \alpha \to \alpha} \to \text{Intro}$$

Head reduction in proof-theoretic terms, which we will refer to as *head proof normalisation* or head PN, is therefore the normalisation of a proof to head normal-form by normalising only the left-most branch of the proof.

Head-neededness for proofs is defined using the concepts of head normal-form and head PN as follows.

Definition 4.5 *A sub-proof* Q *with conclusion* $\Gamma \vdash \alpha$ *of a proof* P *of* $\Lambda \vdash \sigma$ *is head-needed iff a descendant of* Q *is normalised in the head PN of* P *to head normal-form.*

The related notion of tail-neededness may also be defined for proofs.

Definition 4.6 *The set of tail-needed sub-proofs is the set of sub-proofs normalised in every proof normalisation sequence from the hnf of a proof P to nf, where the hnf has been found by a head PN sequence.*

Using the definitions of descendants, and head and tail neededness, the definitions of *head packets* and *tail packets* are exactly as in Section 3.1, using proof normalisation instead of reduction.

So far, we have couched our definitions in terms of sub-proofs which can be normalised. However, just as neededness of redexes was generalised to neededness of terms in the λ-calculus, we can extend the proof-theoretic definition of neededness to sub-proofs in general by means of a transliteration of Definition 3.6.

4.3 Inferring usage

The previous section was essentially a restatement of the results of Section 3 in proof-theoretic terms, under the authority of the Curry-Howard Isomorphism. Hence we can expect that a proof-theoretic definition of usage will similarly follow from that put forward in Section 3.3.

To denote the usage of hypotheses in a sequent, we introduce the following notation.

Notation 4.1 *Say* $\Gamma, |\alpha|_i \vdash \beta$ *to represent the use of hypothesis* α i *times* ($i \in \mathcal{U}$) *in the normalisation of the proof of* β*.*

In other words, $|\alpha|_i$ denotes the fact that any proof of the hypothesis α will be head and tail needed during normalisation such that its head and tail packets will fall into the ranges specified for i in Table 1.

The question of how to infer usage of hypotheses is answered by simple induction over the structure of proofs. By tautology, the Axiom uses its hypothesis once, i.e. we have

$$\dfrac{}{|\alpha|_1 \vdash \alpha}\text{Axiom}$$

while the Contraction rule practices set union over the ranges of head and tail packet sizes defining the size of the use parcels of the two hypotheses contracted (derived as a consequence from Lemma

3.3, represented here by addition of the use values, i.e.

$$\frac{\Gamma, |\alpha|_i, |\alpha|_j \vdash \beta}{\Gamma, |\alpha|_{i+j} \vdash \beta}\text{ Contraction}$$

Weakening introduces parcels into the left-hand side of the sequent of size 0. The Abstraction rule, from a hypothesis parcel of $|\alpha|_i$ used in the inference of a proposition β creates an implication $\alpha \xrightarrow{i} \beta$. Finally, Application uses the result of Lemma 3.2 to infer the size of the use parcels of those hypotheses used in deducing the argument to the implication, i.e. we have

$$\frac{\Gamma \vdash \alpha \xrightarrow{i} \beta \quad \Delta \vdash \alpha}{\Gamma, \Delta^{*i} \vdash \beta}\text{ Application}$$

(where Δ^{*i} indicates that the use parcels of each of the hypotheses in Δ are multiplied by i). The operations $+$ and $*$ are as defined for use values in Section 3.3. Their definition ensures that the sizes of the use parcels inferred by the Contraction and App rules will be optimal, in the sense that the result of $+$ and $*$ over use parcel sizes will be the least point in the $\mathcal{U}$ domain.

5 A resource-aware type system

In this section, we use the extended logic of the previous section to develop a resource-aware type system for the λ-calculus.

The essential difference between the resource-aware logic and the type system presented here is that the type system has only three rules, one for each of the forms of λ-term, i.e. variables, abstractions and applications. The type system is presented in Appendix B. (Note that the operations of both $+$ and $*$ have been extended to bases. $\Gamma \cup \Delta$, for example, contracts common hypotheses between Γ and Δ.)

In the type system, the structural rules of Contraction and Weakening, although not present explicitly, are subsumed into the two rules for variables and applications, Var and App, respectively. (Exchange becomes entirely implicit in our assumption of the base of a sequent as being an unordered set.)

6 Conclusions and related work

We have demonstrated that the syntactic notion of needed redexes can be generalised to define the semantic property of resource use in functions. As part of this work, we introduced a variant of neededness, tail-neededness, and defined packets of needed descendants, in order to measure the degree to which redexes were needed. These results were generalised within a term-model. Further work will include looking at definitions of neededness and usage that are independent of reduction order.

We also applied neededness to proofs, to demonstrate an equivalent definition of resource use at the level of types. The role of the structural rules of Contraction and Weakening in determining usage were discussed. We speculate that the description of the role of these rules, especially of Contraction, together with our definition of resource use, may form the basis for a correspondence with Intuitionistic Linear Logic [Gir87] [Abr92].

On the basis of the above results, a resource-aware type system was briefly presented.

Related to the work described in this paper are the resource-aware type systems of Wright [Wri92] and Baker-Finch [Bak92]. ([WBF93] is a collaborative work.) Wright's thesis develops a type system involving a boolean algebra of function types to capture strictness and absence. An outline of a type system capable of inferring precise sharing information, where the function annotations are taken from the set of Natural numbers, is given, and this is drawn in greater detail in Baker-Finch's work. Unfortunately, as they point out, any algorithm implementing this system would be required to unify the algebraic usage expressions that annotate function types, an unsolvable problem. The finiteness of our domain of use values should ensure that an implementation of our work should not be similarly afflicted.

In both works, the underlying semantic framework used to demonstrate soundness and completeness of the type systems, employs the idea of needed redexes. In [Bak92], the number of needed descendants of a redex within a term are counted to provide the semantics of function types and to support the proofs of soundness and completeness of the type system. We acknowledge the influence of this work upon our own, in leading us to consider neededness to measure usage.

Also of relevance, as mentioned in [Wri92] and [Bak92], is the usage interval analysis in [Ses91], in which usage intervals are defined using the values *Zero*, *One* and *Many*. A usage interval gives the lower and upper bounds within which the actual use of a function argument may fall; for example, the interval $[Zero, One]$ means that a function argument may not be used but will at most be used once. In the context of our use values, we obtain the following equivalences

$$\begin{array}{lcl} [Zero, Zero] & \equiv & 0 \\ [Zero, One] & \equiv & 0+ \\ [Zero, Many] & \equiv & 0+ \\ [One, One] & \equiv & 1 \\ [One, Many] & \equiv & 1+ \\ [Many, Many] & \equiv & 1+ \end{array}$$

It is apparent that Sestoft's usage intervals provide greater detail than our usage values; for example, there is no equivalent usage value for the interval $[Zero, One]$, which may be interpreted as non-strict but unshared.

Also related to our type system is Bierman's work [Bie92] on a type system based on Bounded Linear Logic [GSS92], which, by appearance is similar to our type system and those of Wright and Baker-Finch, although the semantic basis of Bierman's type system is not known. (Baker-Finch outlines an extension of his type system to Bierman's lattice of annotations.)

References

[Abr92] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 1992. Special Issue on the 1990 Workshop on Math. Found. Prog. Semantics. To appear.

[AH87] S. Abramsky and C. Hankin, editors. *Abstract Interpretation of Declarative Languages.* Ellis Horwood Ltd., 1987.

[Arg91] G. Argo. Lifetime analysis. In *Functional Programming, Glasgow 1990.* Springer-Verlag, 1991.

[Bak92] C. Baker-Finch. Relevance and contraction : A logical basis for strictness and sharing analysis. Technical report, University of Canberra, July 1992.

[Bar84] H. P. Barendregt. *The Lambda Calculus : Its Syntax and Semantics.* Studies in Logic and the Foundations of Mathematics. North Holland, 1984.

[BHA86] G. L. Burn, C. L. Hankin, and S. Abramsky. Strictness analysis for higher-order functions. *Science of Computer Programming*, 7:249–278, 1986.

[Bie92] G. Bierman. Type systems, linearity and functional languages, 1992. Slides from talk given at CLICS Workshop, Aarhus University, Denmark.

[BKKS86] H. P. Barendregt, J. R. Kennaway, J. W. Klop, and M. R. Sleep. Needed reduction and spine strategies for the lambda calculus. Technical Report CS-R8621, Centrum voor Wiskunde en Informatica, 1986.

[Blo89] A. Bloss. Update analysis and the efficient implementation of functional aggregates. In *FPCA '89 Conference Proceedings*, pages 26–38. ACM Press, 1989.

[Bur92] G. L. Burn. A logical framework for program analysis. Technical report, Dept. of Computing, Imperial College, 1992.

[CPJ85] C.D. Clack and S.L. Peyton Jones. Strictness analysis — a practical approach. In *Proceedings of FPCA Conference*, pages 35–49. ACM, Springer Verlag, September 1985. LNCS 201.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir89] J.-Y. Girard. *Proofs and Types.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1989.

[GSS92] J.-Y. Girard, A. Scedrov, and P.J. Scott. Bounded linear logic: A modular approach to polynomial time computability. *Theoretical Computer Science*, 97:1–66, 1992.

[Hin83] R. Hindley. The completeness theorem for typing λ-terms. *Theoretical Computer Science*, 22:1–17, 1983.

[How80] W. A. Howard. The formuale-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, 1980.

[HS86] J. R. Hindley and J. P. Seldin. *Introduction to Combinators and λ-Calculus*, volume 1 of *London Mathematical Society Student Texts.* Cambridge University Press, 1986.

[Hug91] S. Hughes. *Static Analysis of Store Use in Functional Programs.* PhD thesis, Dept. of Computing, Imperial College, London, 1991.

[Jen91] T. P. Jensen. Strictness analysis in logical form. In J. Hughes, editor, *Proc. of 5th ACM Conference on Functional programmig Languages and Computer Architecture.* LNCS vol. 523. Springer Verlag, 1991.

[Jen92] T. P. Jensen. *Abstract Interpretation In Logical Form.* PhD thesis, Imperial College, London, November 1992.

[Klo80] J. W. Klop. *Combinatory Reduction Systems.* Mathematical Centre Tracts 127, 1980.

[KM89] T. Kuo and P. Mishra. Strictness analysis: A new perspective based on type inference. In *Proc. of 4th ACM Conference on Functional programmig Languages and Computer Architecture.* ACM Press, 1989.

[Lin92] P. D. Lincoln. *Computational Aspects of Linear Logic.* PhD thesis, Stanford University, August 1992.

[Mac91] I. Mackie. Lilac. Master's thesis, Imperial College, Dept of Computing, September 1991.

[Ses91] P. Sestoft. *Analysis and Efficient Implementation of Functional Programs.* PhD thesis, Department of Computer Science, University of Copenhagen, October 1991.

[Wad91] P. Wadler. Is there a use for linear logic? In *Proc. ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, 1991.

[WBF93] D. A. Wright and C. A. Baker-Finch. Usage analysis with natural reduction types. In *Workshop on Static Analysis*, 1993.

[Wri92] D. A. Wright. *Reduction Types and Intensionality in the Lambda-Calculus*. PhD thesis, University of Tasmania, 1992.

A The implicative fragment of Intuitionistic Logic

Axiom

$$\frac{}{\alpha \vdash \alpha}\ \text{Axiom}$$

Structural rules

$$\frac{\Gamma, \alpha, \alpha \vdash \beta}{\Gamma, \alpha \vdash \beta}\ \text{Contraction}$$

$$\frac{\Gamma \vdash \beta}{\Gamma, \alpha \vdash \beta}\ \text{Weakening}$$

Logical rules

$$\frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \rightarrow \beta} \rightarrow -\text{Intro}$$

$$\frac{\Gamma \vdash \alpha \rightarrow \beta \quad \Delta \vdash \alpha}{\Gamma, \Delta \vdash \beta} \rightarrow -\text{Elim}$$

B A Resource-Aware Type System

$$\frac{}{\Gamma_0, |\mathtt{x} : \alpha|_1 \vdash \mathtt{x} : \alpha}\ \text{Var}$$

$$\frac{\gamma, |\mathtt{x} : \alpha|_i \vdash \mathtt{E} : \beta}{\Gamma \vdash \lambda \mathtt{x}.\mathtt{E} : \alpha \xrightarrow{i} \beta}\ \text{Abs}$$

$$\frac{\Gamma \vdash \mathtt{E} : \alpha \xrightarrow{i} \beta \quad \Delta \vdash \mathtt{F} : \alpha}{\Gamma + \Delta^{*i} \vdash \mathtt{EF} : \beta}\ \text{App}$$