Short Presentation: A Calculus for Resource Relationships

Robert Atkey
LFCS, Divison of Informatics
University of Edinburgh
Mayfield Road
Edinburgh EH9 3JZ, UK
bob.atkey@ed.ac.uk

1. INTRODUCTION

We describe an extension of Bunched Typing, or the $\alpha\lambda$-calculus, as described by O'Hearn [2], intended to increase the flexibility of the system and to suggest further possible routes for investigation of bunched type systems.

The system, called λ_{sep} extends the two binary context formers of the $\alpha\lambda$-calculus to n-place context formers with arbitrary *separation* relations between their members. These relations express pairwise separation constraints on the resources used by the members of the context. The system then allows a slight distinction between the combination of resources and the expression of constraints between resources. This system can express certain constraints that the $\alpha\lambda$-calculus cannot, and can express other constraints in a clearer way. This system is described more fully, with categorical and functor category semantics, in [1].

This research was supported by the MRG project (IST-2001-33149) which is funded by the EC under the FET proactive initiative on Global Computing.

2. SEPARATION TYPING

The logic of Bunched Implications [3] has generated much interest recently, mainly due to its spatial interpretation. One can read the new $A{*}B$ connective of the logic as making an assertion that the resources used by the formulas A and B are separate. BI also has the related implication $A \mathrel{-\!*} B$, which intuitively means that the proof of any A we apply to this implication must use separate resources from the proof of the implication itself. The Curry-Howard analogue of BI is the $\alpha\lambda$-calculus [2], a typed λ-calculus which uses the connectives of BI to make assertions about the resources uses by its objects. The normal product type $A \times B$ allows its components to share resources, while the product $A * B$ does not.

The system presented here, λ_{sep}, is based on the observation that we often consider the two operations of combining resources and distinguishing separate resources as distinct. For example, if we regard our resources as sets of memory of locations then we have the two operations of taking the union of two sets and the predicate of disjointness to check for separation.

Another example is where we can consider resources to be sets of security tokens. We combine two sets of security tokens by taking the union; and relate two security levels with an "information flow" predicate, allowing information to flow from one object to another if the first object has a subset of the security tokens of the second.

We incorporate this notion in the type system by contexts which have an attached relation between their members. Thus a context of the form:

$$[1\#2](a : A, b : B, c : C)$$

would describe a situation where we have access to three variables a, b and c where a and b are guaranteed to be using separate resources. The juxtaposition of the members models the combination and the relation models the relationships between the resources used by the terms.

In general contexts may have arbitrary relations of the appropriate size attached to them. These may be forced to be symmetric in the case of modeling the separation of resources or some other symmetric property of resources. For the purposes of the formal rules given here we will only be considering the case of modeling the separation of resources.

We may nest contexts, similar to the bunches of the $\alpha\lambda$-calculus:

$$[1\#2]([1\#2](a : A, b : B), c : C)$$

This context describes a situation where we have access to a and b which are separate from each other, and are also separate from c. This same situation may also be described by the following context:

$$[1\#2, 1\#3, 2\#3](a : A, b : B, c : C)$$

Here we have substituted the inner context into the outer one, and updated the outer separation relation accordingly. In the type system we regard such pairs of contexts as equal via the equivalence relation $\equiv$. This formalizes the observation that an object which is separate from a collection of things is also separate from them individually. We also regard two contexts which are permutations of one another as equal.

Note that we can recover the bunches of the $\alpha\lambda$-calculus by restricting to binary contexts. Then '$\equiv$' becomes a complicated way to state the associativity and exchange rules.

Structural rules

$$\frac{}{x : A \vdash x : A}\ (\textsc{Id}) \qquad \frac{\Gamma([](\Delta, \Delta')) \vdash e : A \qquad \Delta \cong \Delta'}{\Gamma(\Delta) \vdash e[\mathrm{id}(\Delta)/\mathrm{id}(\Delta')] : A}\ (\textsc{Contraction}) \qquad \frac{\Gamma \vdash e : A \qquad \Gamma \equiv \Gamma'}{\Gamma' \vdash e : A}\ (\textsc{Equiv})$$

$$\frac{S(\overrightarrow{\Gamma}) \vdash e : A \qquad S \subseteq S'}{S'(\overrightarrow{\Gamma}) \vdash e : A}\ (\text{S-}\textsc{Weaken}) \qquad \frac{\Gamma([]_0()) \vdash e : A}{\Gamma(\Delta) \vdash e : A}\ (\textsc{Weaken})$$

Connective rules

$$\frac{\Gamma_1 \vdash e_1 : A_1 \qquad \ldots \qquad \Gamma_n \vdash e_n : A_n}{S(\Gamma_1, \ldots, \Gamma_n) \vdash S(e_1, \ldots, e_n) : S(A_1, \ldots, A_n)}\ (S\text{-I}) \qquad \frac{\Gamma \vdash e_1 : S(A_1, \ldots, A_n) \qquad \Delta(S(x_1 : A_1, \ldots, x_n : A_n)) \vdash e_2 : B}{\Delta(\Gamma) \vdash \text{let } S(x_1, \ldots, x_n) = e_1 \text{ in } e_2 : B}\ (S\text{-E})$$

$$\frac{S(\Gamma, x_1 : A_1, \ldots, x_n : A_n) \vdash e : B}{\Gamma \vdash \lambda^S(x_1, \ldots, x_n).e : A_1, \ldots, A_n \xrightarrow{S} B}\ (\to\text{-I}) \qquad \frac{\Gamma \vdash f : A_1, \ldots, A_n \xrightarrow{S} B \qquad \text{for } 1 \leq i \leq n. \quad \Delta_i \vdash a_i : A_i}{S(\Gamma, \Delta_1, \ldots, \Delta_n) \vdash f@_S(a_1, \ldots, a_n) : B}\ (\to\text{-E})$$

Figure 1: The rules of λ_{sep}

2.1 The Typing Rules

The typing rules are shown in figure 1. The first collection of rules covers the structural rules of the system. It is here that we model the properties of resource separation and its interaction with resource combination.

The rule Contraction allows the treatment of two nested contexts as one only if the construction of e does not rely on them referring to separate resources, hence the empty relation $[]$ between Δ and Δ'. The relation $\Delta \cong \Delta'$ ensures that Δ and Δ' are identical up to renaming of identifiers.

The Equiv rule replaces contexts with equivalent ones, as described above. The S-Weaken rule allows the addition of constraints on the context, and the Weaken rule allows the addition of extra variables to the context.

The connective rules essentially mirror the structure of the contexts to the right hand side of the turnstile. The S-I and S-E rules introduce and eliminate tuple types, one for each possible separation relation. Note that the tuple type (and context) $[]_0()$ acts as a unit type and the type $[]_1(A)$ acts the same as A. The $\to$-I and $\to$-E rules introduce and eliminate multi argument functions. The multiple arguments are necessary to retain the correct relationships between the abstracted variables and the remaining context.

2.2 Example

Consider a set of primitives for constructing jobs for processing. Each of these jobs consists of two items of data, which are operated on in parallel; they must occupy separate regions of memory (to allow for temporary in-place mutation, for example). This constraint can be captured by a job construction operation:

$$\texttt{consJob} : [1\#2](D, D) \to J$$

This construction is also possible in the $\alpha\lambda$-calculus.

Now consider a collection of such jobs to be run in sequence. Since they are to be run in sequence it does not matter if any of the jobs overlap in memory. A collection of three such jobs, over four items of data, can be represented as:

$$(\texttt{consJob}(\texttt{a}, \texttt{b}), \texttt{consJob}(\texttt{b}, \texttt{c}), \texttt{consJob}(\texttt{c}, \texttt{d})) : [](J, J, J)$$

Now a context for this term should represent the constraints here as accurately as possible; it should prohibit sharing where required, but allow sharing as often as possible. In λ_{sep} the context can be given as:

$$[\texttt{a}\#\texttt{b}, \texttt{b}\#\texttt{c}, \texttt{c}\#\texttt{d}](\texttt{a} : D, \texttt{b} : D, \texttt{c} : D, \texttt{d} : D)$$

Here the only constraints are between members of the context whose separation is forced by the construction of the term. The affine $\alpha\lambda$-calculus cannot express this configuration. The restriction to binary combinations for expressing separation forces a context where there is extraneous separation enforced. One can get close using a context such as (where ';' represents possible sharing, and ',' no sharing):

$$((\texttt{a} : D; \texttt{d} : D), \texttt{b} : D, \texttt{c} : D)$$

However, this requires that `a` and `c` be separate, whereas λ_{sep} does not require this. We also claim that the scheme of separating the separation constraints from the body of the context allows for a clearer and more intuitive expression of constraints.

3. CONCLUSIONS

While the gains in expressivity over the $\alpha\lambda$-calculus may not be particularly impressive, we believe that the calculus described here acts as an interesting base for further investigation of λ-calculi with bunched structure in their typing contexts.

In particular, variations of the system with non-symmetric relations, or with more restricted structural rules – for example removing Weaken – may offer opportunities for larger distinctions between λ_{sep} and the $\alpha\lambda$-calculus. Also, it allows, in our opinion, a cleaner, more intuitive expression of resource constraints.

4. REFERENCES

[1] R. Atkey. A λ-Calculus For Resource Separation. WWW: `http://www.dcs.ed.ac.uk/home/roba`, 2003.

[2] P. W. O'Hearn. On bunched typing. *Journal of Functional Programming*, 13(4):747–796, 2003.

[3] D. J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, 2002.