Counting Successes: Effects and Transformations for Non-Deterministic Programs

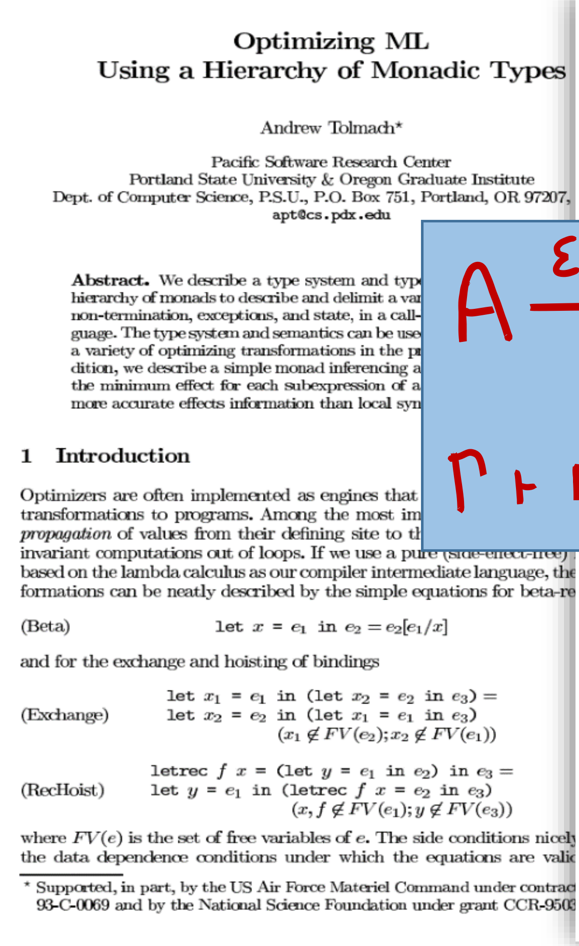
Nick Benton (Microsoft)
Andrew Kennedy (Facebook)
Martin Hofmann (LMU)
Vivek Nigam (UFPB)

Monads, Effect Systems

- Type-and-effect systems (Gifford & Lucassen, 1986)
  - $\Gamma \vdash M : A, \varepsilon$
  - $A \; ::= \cdots \mid A \xrightarrow{\varepsilon} B$
- Monads and computational metalanguage (Moggi, 1989)
  - $X ::= \cdots \mid TX$
  - $(A \to B)^* = A^* \to T(B^*)$
  - $(\Gamma \vdash M : A)^* = \Gamma^* \vdash M^* : T(A^*)$
  - `let` and `val` constructs, nice equational theory

1998: something in the air

TIC



Optimizing ML
Using a Hierarchy of Monadic Types

Andrew Tolmach*

Pacific Software Research Center
Portland State University & Oregon Graduate Institute
Dept. of Computer Science, P.S.U., P.O. Box 751, Portland, OR 97207,
apt@cs.pdx.edu

Abstract. We describe a type system and typ
hierarchy of monads to describe and delimit a var
non-termination, exceptions, and state, in a call-
guage. The type system and semantics can be use
a variety of optimizing transformations in the pr
dition, we describe a simple monad inferencing a
the minimum effect for each subexpression of a
more accurate effects information than local syn

1 Introduction

Optimizers are often implemented as engines that
transformations to programs. Among the most im
propagation of values from their defining site to th
invariant computations out of loops. If we use a pure (side-effect-free)
based on the lambda calculus as our compiler intermediate language, the
formations can be neatly described by the simple equations for beta-re

(Beta) $\texttt{let}\ x = e_1\ \texttt{in}\ e_2 = e_2[e_1/x]$

and for the exchange and hoisting of bindings

$$\text{(Exchange)}\quad \begin{array}{l} \texttt{let}\ x_1 = e_1\ \texttt{in}\ (\texttt{let}\ x_2 = e_2\ \texttt{in}\ e_3) = \\ \texttt{let}\ x_2 = e_2\ \texttt{in}\ (\texttt{let}\ x_1 = e_1\ \texttt{in}\ e_3) \\ \qquad (x_1 \notin FV(e_2); x_2 \notin FV(e_1)) \end{array}$$

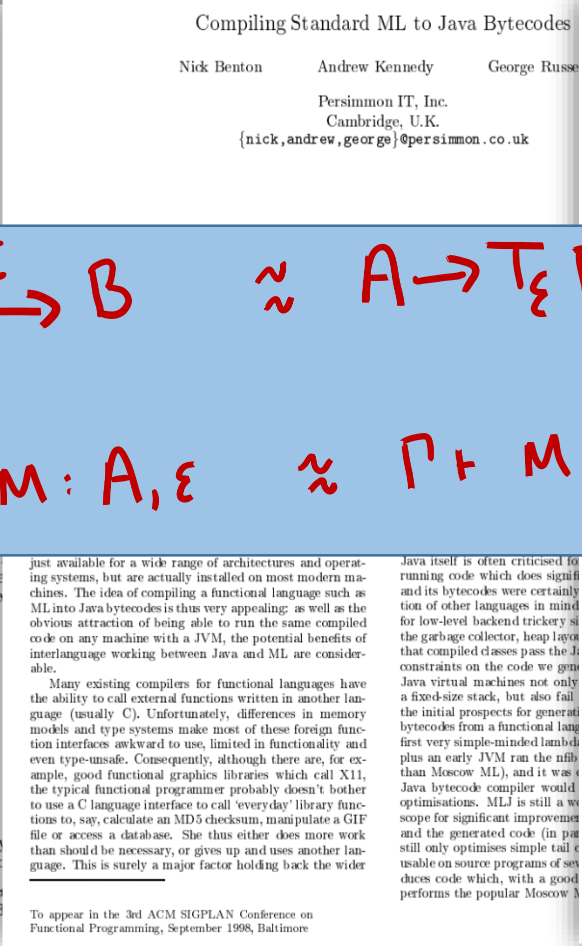
$$\text{(RecHoist)}\quad \begin{array}{l} \texttt{letrec}\ f\ x = (\texttt{let}\ y = e_1\ \texttt{in}\ e_2)\ \texttt{in}\ e_3 = \\ \texttt{let}\ y = e_1\ \texttt{in}\ (\texttt{letrec}\ f\ x = e_2\ \texttt{in}\ e_3) \\ \qquad (x, f \notin FV(e_1); y \notin FV(e_3)) \end{array}$$

where $FV(e)$ is the set of free variables of e. The side conditions nicely
the data dependence conditions under which the equations are valic

---

* Supported, in part, by the US Air Force Materiel Command under contrac
93-C-0069 and by the National Science Foundation under grant CCR-9503

ICFP



Compiling Standard ML to Java Bytecodes

Nick Benton Andrew Kennedy George Russe

Persimmon IT, Inc.
Cambridge, U.K.
{nick,andrew,george}@persimmon.co.uk

just available for a wide range of architectures and operating systems, but are actually installed on most modern machines. The idea of compiling a functional language such as ML into Java bytecodes is thus very appealing: as well as the obvious attraction of being able to run the same compiled code on any machine with a JVM, the potential benefits of interlanguage working between Java and ML are considerable.

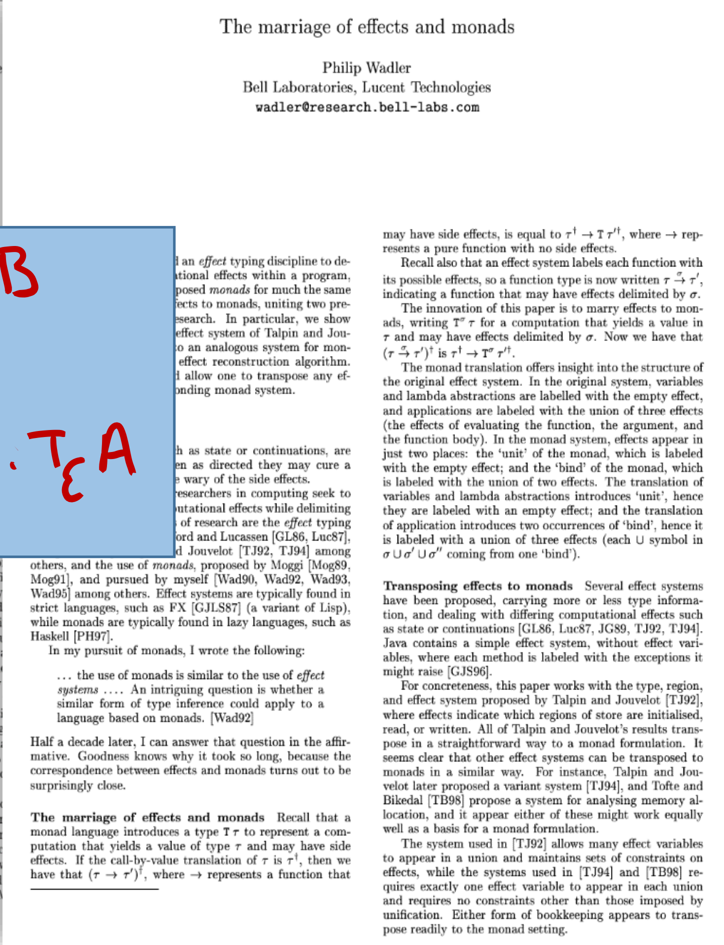
Many existing compilers for functional languages have the ability to call external functions written in another language (usually C). Unfortunately, differences in memory models and type systems make most of these foreign function interfaces awkward to use, limited in functionality and even type-unsafe. Consequently, although there are, for example, good functional graphics libraries which call X11, the typical functional programmer probably doesn't bother to use a C language interface to call 'everyday' library functions to, say, calculate an MD5 checksum, manipulate a GIF file or access a database. She thus either does more work than should be necessary, or gives up and uses another language. This is surely a major factor holding back the wider

Java itself is often criticised fo
running code which does signifi
and its bytecodes were certainly
tion of other languages in mind
for low-level backend trickery si
the garbage collector, heap layou
that compiled classes pass the Ja
constraints on the code we gene
Java virtual machines not only
a fixed-size stack, but also fail
the initial prospects for generati
bytecodes from a functional lang
first very simple-minded lambda
plus an early JVM ran the nfib
than Moscow ML), and it was
Java bytecode compiler would
optimisations. MLJ is still a wo
scope for significant improvemen
and the generated code (in par
still only optimises simple tail c
usable on source programs of sev
duces code which, with a good
performs the popular Moscow M

To appear in the 3rd ACM SIGPLAN Conference on Functional Programming, September 1998, Baltimore

---

ICFP



The marriage of effects and monads

Philip Wadler
Bell Laboratories, Lucent Technologies
wadler@research.bell-labs.com

l an *effect* typing discipline to de-
ational effects within a program,
posed *monads* for much the same
fects to monads, uniting two pre-
esearch. In particular, we show
effect system of Talpin and Jou-
o an analogous system for mon-
effect reconstruction algorithm.
l allow one to transpose any ef-
onding monad system.

h as state or continuations, are
en as directed they may cure a
e wary of the side effects.
researchers in computing seek to
utational effects while delimiting
of research are the *effect* typing
ord and Lucassen [GL86, Luc87],
d Jouvelot [TJ92, TJ94] among
others, and the use of *monads*, proposed by Moggi [Mog89, Mog91], and pursued by myself [Wad90, Wad92, Wad93, Wad95] among others. Effect systems are typically found in strict languages, such as FX [GJLS87] (a variant of Lisp), while monads are typically found in lazy languages, such as Haskell [PH97].

In my pursuit of monads, I wrote the following:

> … the use of monads is similar to the use of *effect systems* …. An intriguing question is whether a similar form of type inference could apply to a language based on monads. [Wad92]

Half a decade later, I can answer that question in the affirmative. Goodness knows why it took so long, because the correspondence between effects and monads turns out to be surprisingly close.

The marriage of effects and monads Recall that a monad language introduces a type $\mathrm{T}\,\tau$ to represent a computation that yields a value of type τ and may have side effects. If the call-by-value translation of τ is $\tau^\dagger$, then we have that $(\tau \to \tau')^\dagger$, where $\to$ represents a function that may have side effects, is equal to $\tau^\dagger \to \mathrm{T}\,\tau'^\dagger$, where $\to$ represents a pure function with no side effects.

Recall also that an effect system labels each function with its possible effects, so a function type is now written $\tau \xrightarrow{\sigma} \tau'$, indicating a function that may have effects delimited by σ.

The innovation of this paper is to marry effects to monads, writing $\mathrm{T}^\sigma\,\tau$ for a computation that yields a value in τ and may have effects delimited by σ. Now we have that $(\tau \xrightarrow{\sigma} \tau')^\dagger$ is $\tau^\dagger \to \mathrm{T}^\sigma\,\tau'^\dagger$.

The monad translation offers insight into the structure of the original effect system. In the original system, variables and lambda abstractions are labelled with the empty effect, and applications are labeled with the union of three effects (the effects of evaluating the function, the argument, and the function body). In the monad system, effects appear in just two places: the 'unit' of the monad, which is labeled with the empty effect; and the 'bind' of the monad, which is labeled with the union of two effects. The translation of variables and lambda abstractions introduces 'unit', hence they are labeled with an empty effect; and the translation of application introduces two occurrences of 'bind', hence it is labeled with a union of three effects (each $\cup$ symbol in $\sigma \cup \sigma' \cup \sigma''$ coming from one 'bind').

Transposing effects to monads Several effect systems have been proposed, carrying more or less type information, and dealing with differing computational effects such as state or continuations [GL86, Luc87, JG89, TJ92, TJ94]. Java contains a simple effect system, without effect variables, where each method is labeled with the exceptions it might raise [GJS96].

For concreteness, this paper works with the type, region, and effect system proposed by Talpin and Jouvelot [TJ92], where effects indicate which regions of store are initialised, read, or written. All of Talpin and Jouvelot's results transpose in a straightforward way to a monad formulation. It seems clear that other effect systems can be transposed to monads in a similar way. For instance, Talpin and Jouvelot later proposed a variant system [TJ94], and Tofte and Bikedal [TB98] propose a system for analysing memory allocation, and it appear either of these might work equally well as a basis for a monad formulation.

The system used in [TJ92] allows many effect variables to appear in a union and maintains sets of constraints on effects, while the systems used in [TJ94] and [TB98] requires exactly one effect variable to appear in each union and requires no constraints other than those imposed by unification. Either form of bookkeeping appears to transpose readily to the monad setting.

$$A \xrightarrow{\varepsilon} B \quad \approx \quad A \to T_\varepsilon B$$

$$\Gamma \vdash M : A, \varepsilon \quad \approx \quad \Gamma \vdash M : T_\varepsilon A$$

Transformations

$$\text{let } x = M \text{ in } N \quad = \quad N$$

if $\quad x \notin fv(N)$

and M doesn't

i) diverge
ii) write the state
iii) throw any exceptions

(it's allowed to read and/or allocate, though).

- HOOTS 1999

Realizability to the rescue

- Reading, Writing & Relations, APLAS 2006
- Could have been called “Optimizing Transformations for Free!”
- Interpret types as binary relations (PERs) over untyped model
  - Or refined types as relations over unrefined typed model
- Soundness of rules: terms related to themselves by interpretation of their types
- Soundness of transformations: different terms related to one another by interpretation of a type
  - $\Gamma \vdash M = M' \colon A$

How to do this for “funny” type systems?

$$[\![T_\epsilon X]\!] \subseteq (S \to S \times [\![UX]\!]) \times (S \to S \times [\![UX]\!])$$

$$[\![T_\epsilon X]\!] = \bigcap_{R \in \mathcal{R}_\epsilon} R \Rightarrow R \times [\![X]\!]$$

$$\mathcal{R}_\epsilon, \mathcal{R}_e \subseteq \mathbb{P}(S \times S)$$

$$\mathcal{R}_\epsilon = \bigcap_{e \in \epsilon} \mathcal{R}_e$$

$$\mathcal{R}_{r_\ell} = \{R \mid \forall (s, s') \in R, s\,\ell = s'\ell\}$$

$$\mathcal{R}_{w_\ell} = \{R \mid \forall (s, s') \in R, n \in \mathbb{Z}. (s[\ell \mapsto n], s'[\ell \mapsto n]) \in R\}$$

$$\frac{\Theta \vdash M : T_{\epsilon_1} X \qquad \Theta, x : X, y : X \vdash N : T_{\epsilon_2} Y}{\Theta \vdash let\ x \Leftarrow M; y \Leftarrow M\ in\ N = let\ x \Leftarrow M\ in\ N[x/y] :\ T_{\epsilon_1 \cup \epsilon_2} Y}\ rds(\epsilon_1) \cap wrs(\epsilon_1) = \emptyset$$

Extensions and variations

- Dynamic allocation (regions, masking)
- Higher-typed store (not entirely successful)
- Abstract locations (proof-relevant logical relations)
- Concurrency
- Exceptions

Non-Determinism

- How to Replace Failure by a List of Successes, Wadler 1985
  - Shows how (lazy) lists can be used to program both computations with errors and logic-programming style backtracking search
  - Now have `MonadPlus` in Haskell (and long-standing debates about what equations should be satisfied)
- Here: refined types for a simple (total) non-deterministic language capturing how many results a computation may return
  - Again, relational semantics gives equations

Base language

- Computational metalanguage with operations

$$\frac{}{\Gamma \vdash \mathtt{fail} : TA} \qquad \frac{\Gamma \vdash M_1 : TA \quad \Gamma \vdash M_2 : TA}{\Gamma \vdash M_1 \,\mathtt{or}\, M_2 : TA}$$

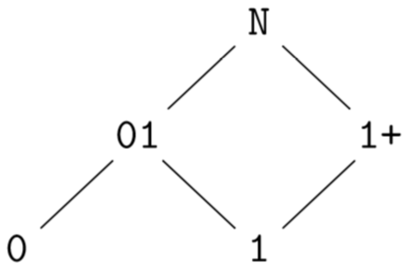
- Sets and functions with $[\![TA]\!] = \mathbb{P}_{fin}([\![A]\!])$

$$\begin{aligned}
[\![\Gamma \vdash \mathtt{val}\, V : TA]\!]\,\rho &= \{[\![\Gamma \vdash V : A]\!]\,\rho\} \\
[\![\Gamma \vdash \mathtt{let}\, x \Leftarrow M \,\mathtt{in}\, N]\!]\,\rho &= \textstyle\bigcup_{v \in [\![\Gamma \vdash M : A]\!]\,\rho} [\![\Gamma, x : A \vdash N : TB]\!]\,(\rho, v) \\
[\![\Gamma \vdash \mathtt{fail} : TA]\!]\,\rho &= \emptyset \\
[\![\Gamma \vdash M_1 \,\mathtt{or}\, M_2 : TA]\!]\,\rho &= ([\![\Gamma \vdash M_1 : TA]\!]\,\rho) \cup ([\![\Gamma \vdash M_1 : TA]\!]\,\rho)
\end{aligned}$$

Effect types

$$X, Y := \texttt{unit} \mid \texttt{int} \mid \texttt{bool} \mid X \times Y \mid X \to T_\varepsilon Y$$
$$\varepsilon \in \{\texttt{0}, \texttt{1}, \texttt{01}, \texttt{1+}, \texttt{N}\}$$

$$\begin{aligned} [\![\texttt{0}]\!] &= \{0\} & [\![\texttt{1+}]\!] &= \{n \mid n \geq 1\} \\ [\![\texttt{1}]\!] &= \{1\} & [\![\texttt{N}]\!] &= \mathbb{N} \\ [\![\texttt{01}]\!] &= \{0, 1\} \end{aligned}$$



Refined types

$$\frac{\Theta \vdash V : X}{\Theta \vdash \mathtt{val}\, V : T_1 X}$$

$$\frac{\Theta \vdash M : T_\varepsilon X \quad \Theta, x : X \vdash N : T_{\varepsilon'} Y}{\Theta \vdash \mathtt{let}\; x \Leftarrow M \;\mathtt{in}\; N : T_{\varepsilon \cdot \varepsilon'} Y}$$

$$\frac{}{\Theta \vdash \mathtt{fail} : T_0 X}$$

$$\frac{\Theta \vdash M_1 : T_{\varepsilon_1} X \quad \Theta \vdash M_2 : T_{\varepsilon_2} X}{\Theta \vdash M_1 \;\mathtt{or}\; M_2 : T_{\varepsilon_1 + \varepsilon_2} X}$$

·	0	1	01	1+	N
0	0	0	0	0	0
1	0	1	01	1+	N
01	0	01	01	N	N
1+	0	1+	N	1+	N
N	0	N	N	N	N

+	0	1	01	1+	N
0	0	1	01	1+	N
1	1	1+	1+	1+	1+
01	01	1+	N	1+	N
1+	1+	1+	1+	1+	1+
N	N	1+	N	1+	N

Semantics

$$[\![X]\!] \subseteq [\![U(X)]\!] \times [\![U(X)]\!]$$

$$[\![\texttt{int}]\!] = \Delta_{\mathbb{Z}} \qquad [\![\texttt{bool}]\!] = \Delta_{\mathbb{B}} \qquad [\![\texttt{unit}]\!] = \Delta_1$$

$$[\![X \times Y]\!] = [\![X]\!] \times [\![Y]\!]$$

$$[\![X \to T_\varepsilon Y]\!] = [\![X]\!] \to [\![T_\varepsilon Y]\!]$$

$$[\![T_\varepsilon X]\!] = \{(S, S') \mid S \sim_X S' \text{ and } |S/[\![X]\!]| \in [\![\varepsilon]\!]\}$$

$where\ S{\sim_X}S' \overset{\text{def}}{=} \forall a \in S, \exists a' \in S', (a, a') \in [\![X]\!]\ and\ v.v.$

$and\ S/[\![X]\!] \overset{\text{def}}{=} \left\{[a]_{[\![X]\!]} \middle| a \in S\right\}$

Results

- Get fundamental theorem, validates all usual equations of metalanguage
- Plus monad-specific, effect *in*dependent laws

Choice:

$$\frac{\Theta \vdash M_1 : T_{\varepsilon_1} X \quad \Theta \vdash M_2 : T_{\varepsilon_2} X}{\Theta \vdash M_1 \,\mathtt{or}\, M_2 = M_2 \,\mathtt{or}\, M_1 : T_{\varepsilon_1 + \varepsilon_2} X}$$

$$\frac{\Theta \vdash M : T_{\varepsilon} X}{\Theta \vdash M \,\mathtt{or}\, M = M : T_{\varepsilon} X} \qquad \frac{\Theta \vdash M : T_{\varepsilon} X}{\Theta \vdash M \,\mathtt{or}\, \mathtt{fail} = M : T_{\varepsilon} X}$$

$$\frac{\Theta \vdash M_1 : T_{\varepsilon_1} X \quad \Theta \vdash M_2 : T_{\varepsilon_2} X \quad \Theta \vdash M_3 : T_{\varepsilon_3} X}{\Theta \vdash M_1 \,\mathtt{or}\, (M_2 \,\mathtt{or}\, M_3) = (M_1 \,\mathtt{or}\, M_2) \,\mathtt{or}\, M_3 : T_{\varepsilon_1 + \varepsilon_2 + \varepsilon_3} X}$$

Commutativity:

$$\frac{\Theta \vdash M : T_{\varepsilon_1} Y \quad \Theta \vdash N : T_{\varepsilon_2} X \quad \Theta, x : X, y : Y \vdash P : T_{\varepsilon_3} Z}{\Theta \vdash \mathtt{let}\, x \Leftarrow M \,\mathtt{in}\, \mathtt{let}\, y \Leftarrow N \,\mathtt{in}\, P = \mathtt{let}\, y \Leftarrow N \,\mathtt{in}\, \mathtt{let}\, x \Leftarrow M \,\mathtt{in}\, P : T_{\varepsilon_1 \cdot \varepsilon_2 \cdot \varepsilon_3} Z}$$

Distribution:

$$\frac{\Theta \vdash M_1 : T_{\varepsilon_1} X \quad \Theta \vdash M_2 : T_{\varepsilon_2} X \quad \Theta, x : X \vdash N : T_{\varepsilon_3} Y}{\Theta \vdash \mathtt{let}\, x \Leftarrow (M_1 \,\mathtt{or}\, M_2) \,\mathtt{in}\, N = (\mathtt{let}\, x \Leftarrow M_1 \,\mathtt{in}\, N) \,\mathtt{or}\, (\mathtt{let}\, x \Leftarrow M_2 \,\mathtt{in}\, N) : T_{(\varepsilon_1 + \varepsilon_2) \cdot \varepsilon_3} Y}$$

Effect-dependent equivalences

Fail:

$$\frac{\Theta \vdash M : T_0 X}{\Theta \vdash M = \texttt{fail} : T_0 X}$$

Dead Computation:

$$\frac{\Theta \vdash M : T_{1+} X \quad \Theta \vdash N : T_\varepsilon Y}{\Theta \vdash \texttt{let}\ x \Leftarrow M\ \texttt{in}\ N = N : T_\varepsilon Y}$$

Duplicated Computation:

$$\frac{\Theta \vdash M : T_{01} X \quad \Theta, x : X, y : X \vdash N : T_\varepsilon Y}{\Theta \vdash \begin{array}{l} \texttt{let}\ x \Leftarrow M\ \texttt{in}\ \texttt{let}\ y \Leftarrow M\ \texttt{in}\ N \\ = \texttt{let}\ x \Leftarrow M\ \texttt{in}\ N[x/y] \end{array} : T_{01 \cdot \varepsilon} Y}$$

Pure Lambda Hoist:

$$\frac{\Theta \vdash M : T_1 Z \quad \Theta, x : X, y : Z \vdash N : T_\varepsilon Y}{\Theta \vdash \begin{array}{l} \texttt{val}\ (\lambda x : U(X).\texttt{let}\ y \Leftarrow M\ \texttt{in}\ N) \\ = \texttt{let}\ y \Leftarrow M\ \texttt{in}\ \texttt{val}\ (\lambda x : U(X).N) \end{array} : T_1(X \to T_\varepsilon Y)}$$

Related

- Kammar & Plotkin POPL 2012
  - General approach using algebraic effects
  - Ours not an instance as refined interpretations not all monads
- Katsumata POPL 2014
  - Monoidal functors from preordered monoid to endofunctors on category of values
  - (Also graded monads – but I just heard about this half an hour ago…)
- Lots of work on cardinality analysis in logic programming
  - Mercury (Henderson et al) uses exactly the same set of cardinalities as us for optimizations

Happy Birthday Phil!
And thanks for all the inspiration

Type-specific equality

For example, if we define

$$f_1 = \lambda g : \texttt{unit} \to T\texttt{int}.\texttt{let}\, x \Leftarrow g\,()\ \texttt{in}\ \texttt{let}\, y \Leftarrow g\,()\ \texttt{in}\ \texttt{val}\, x + y$$
$$f_2 = \lambda g : \texttt{unit} \to T\texttt{int}.\texttt{let}\, x \Leftarrow g\,()\ \texttt{in}\ \texttt{val}\, x + x$$

then we have $\vdash f_1 = f_2 : (\texttt{unit} \to T_{\texttt{01}}\texttt{int}) \to T_{\texttt{01}}\texttt{int}$ and hence, for example,

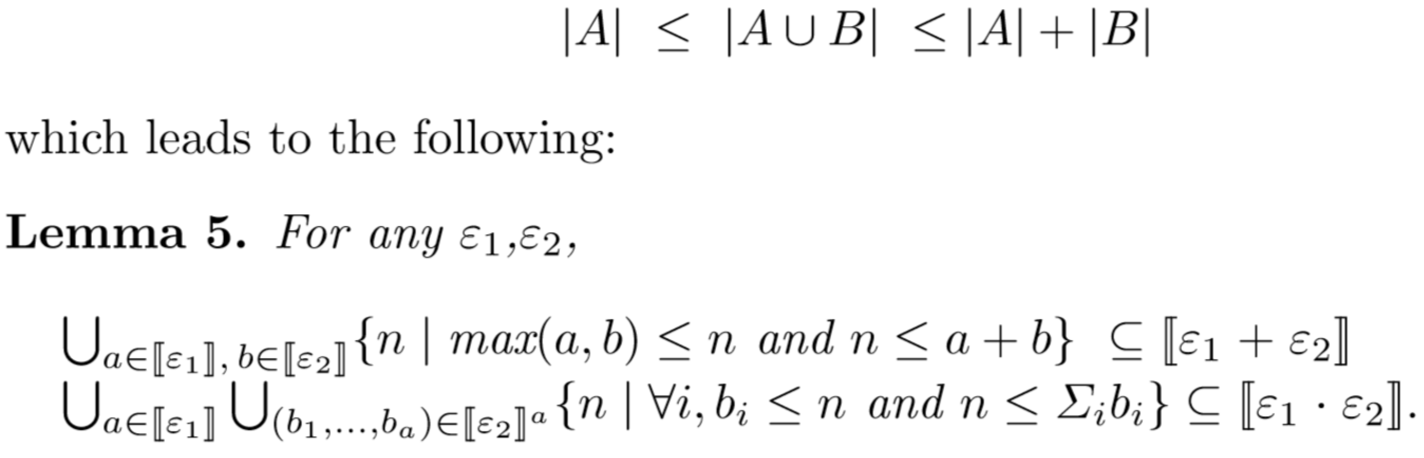
$$\vdash (\texttt{val}\ f_1)\ \texttt{or}\ (\texttt{val}\ f_2) \ = \ \texttt{val}\ f_2 : T_1((\texttt{unit} \to T_{\texttt{01}}\texttt{int}) \to T_{\texttt{01}}\texttt{int}).$$

Note that the notion of equivalence really is type-specific. We have

$$\nvdash f_1 = f_2 : (\texttt{unit} \to T_{\texttt{N}}\texttt{int}) \to T_{\texttt{N}}\texttt{int}$$

and that equivalence indeed does not hold in the semantics, even though both f_1 and f_2 are related to themselves at (i.e. have) that type.

Correctness condition for operators on effect annotations



$$|A| \ \le \ |A \cup B| \ \le |A| + |B|$$

which leads to the following:

Lemma 5. *For any* ε_1,ε_2,

$$\begin{array}{l}\bigcup_{a\in[\![\varepsilon_1]\!],\, b\in[\![\varepsilon_2]\!]}\{n \mid max(a,b) \le n \ \ and \ n \le a+b\} \ \subseteq [\![\varepsilon_1+\varepsilon_2]\!] \\ \bigcup_{a\in[\![\varepsilon_1]\!]}\bigcup_{(b_1,\ldots,b_a)\in[\![\varepsilon_2]\!]^a}\{n \mid \forall i, b_i \le n \ \ and \ n \le \Sigma_i b_i\} \subseteq [\![\varepsilon_1 \cdot \varepsilon_2]\!].\end{array}$$