Electronic Notes in Theoretical Computer Science 26 (1999)
URL: http://www.elsevier.nl/locate/entcs/volume26.html 18 pages

Monads, Effects and Transformations

Nick Benton and Andrew Kennedy

Microsoft Research Ltd., Cambridge, U.K.
{nick,akenn}@microsoft.com

Abstract

We define a typed compiler intermediate language, MIL-lite, which incorporates computational types refined with effect information. We characterise MIL-lite observational congruence by using Howe's method to prove a ciu theorem for the language in terms of a termination predicate defined directly on the term. We then define a logical predicate which captures an observable version of the intended meaning of each of our effect annotations. Having proved the fundamental theorem for this predicate, we use it with the ciu theorem to validate a number of effect-based transformations performed by the MLj compiler for Standard ML.

1 Introduction

The evaluation of an expression in a language such as Standard ML may do a number of things as well as (or instead of) producing a value. These possible *computational effects* include failing to terminate, reading, writing or allocating mutable reference cells, engaging in I/O and raising exceptions. The presence of effects in the language invalidates, in the general case, most of the straightforward equational laws which hold in 'purer' typed lambda calculi. Effects can also complicate the type system, as shown by the uneasy coexistence of references and polymorphism in ML: the SML'97 definition restricts polymorphic generalisation to non-side-effecting expressions by requiring them to be syntactic values; this 'value restriction' replaces the more complex system of imperative types used in SML'90 to avoid type unsoundness.

There are several reasons for wishing to be able to infer automatically (a conservative approximation to) the set of effects a given expression may have. In early work, the justification was often the control of polymorphic generalisation, though for us, the most important application of effect inference is in optimising compilers for ML-like languages. Even a seemingly trivial rewrite, such as the dead-code elimination

$$\texttt{let val x =}\ M_1\ \texttt{in}\ M_2\ \texttt{end} \rightsquigarrow M_2 \qquad (\texttt{x} \notin FV(M_2))$$

is generally only valid if the evaluation of M_1 doesn't diverge, perform I/O, update the state or throw an exception.

There are a number of existing effect analyses for CBV languages, mostly based on a style of non-standard type inference developed by Gifford, Lucassen, Jouvelot and Talpin, amongst others [3,14]. In these systems, the judgement $\Gamma \vdash M : \sigma, \varepsilon$ means that under assumptions Γ, expression M has type σ and effect ε. Function types are also annotated with their 'latent effect', which is the effect which will occur when the function is applied. Thus the rule for abstraction is typically

$$\frac{\Gamma, x : \tau \vdash M : \sigma, \varepsilon}{\Gamma \vdash (\lambda x : \tau . M) : \tau \xrightarrow{\varepsilon} \sigma, \emptyset}$$

because the λ-abstraction itself is a value, and so has no immediate effect ($\emptyset$) but will have effect ε when it is applied.

In the semantics community, meanwhile, there has been much work using Moggi's technique of structuring the denotational semantics of languages with 'impure' features by using monads [9]. Moggi introduced a denotational metalanguage with a type system which distinguishes *computations* (which may have effects) from *values* (which don't). There are adequate translations of both CBV and CBN lambda calculi into this *computational lambda calculus*, and a number of researchers have suggested that Moggi's metalanguage could form the basis for a useful compiler intermediate language which could express, for example, strictness and boxing-based transformations and possibly even be a common target for both strict and lazy languages [1,6].

More recently, the fairly natural idea that effect analyses could be rephrased in terms of a refined monadic type system has received attention from Tolmach [15] and Wadler [16], and has been implemented by the present authors, whose SML-to-Java bytecode compiler, MLj, is built around an intermediate language with effect-specific monadic types [2]. Moggi's CBV translation maps a source language judgement $\{x_i : \sigma_i\} \vdash M : \sigma$ into a metalanguage judgement $\{x_i : \sigma_i^*\} \vdash M^* : T(\sigma^*)$ where T is the computation type constructor and, for example, $\mathtt{int}^* = \mathbb{Z}$ and $(\sigma \to \tau)^* = \sigma^* \to T(\tau^*)$. At first sight, the observation that effect type systems may be rephrased in the monadic framework by annotating the computation type constructor seems to be a triviality: the places effect annotations appear correspond exactly to the places where the T constructor is applied in the CBV translation. But there are both technical and 'philosophical' differences between the two approaches.

The monadic style takes the distinction between computations and values more seriously. Computation types have the same status as other types, with their own introduction and elimination rules and associated term-forming operations, rather than just being ad-hoc annotations on source language types. This makes both evaluation order and the decomposition of the CBV arrow into a 'pure' arrow and a computation type more explicit and gives a natural language for expressing the optimisations one wishes to perform as a result of the analysis. It also has the small advantage of localising the combination of effects into one type rule. But the biggest difference is that the computational lambda calculus has a well-behaved equational theory, and it is this which we need to justify optimisating transformations, rather than just a static analysis.

This paper investigates the (in)equational theory of a simplified fragment of MIL, the monadic intermediate language used in the MLj compiler. This fragment, MIL-lite, lacks some significant features of MIL (polymorphism, higher-type refer-

ences and recursive types) but is far from trivial – combining higher-order functions, recursion, exceptions, subtyping and dynamically allocated state.

2 The MIL-lite Language

2.1 Types and terms

As MIL-lite is a compiler intermediate language for which we first give an operational semantics and then *derive* an equational theory, there are a couple of design differences between it and Moggi's equational metalanguage. The first is that types are stratified into value types (ranged over by τ) and computation types (ranged over by γ); we will have no need of computations of computations. The second difference is that the distinction between computations and values is alarmingly syntactic: the only expressions of value types are normal forms. Given a countable set $\mathbb{E}$ of exception names, MIL-lite types are defined by

$$\tau ::= \mathsf{unit} \mid \mathsf{int} \mid \mathsf{intref} \mid \tau \times \tau \mid \tau + \tau \mid \tau \to \gamma$$
$$\gamma ::= \mathbf{T}_\varepsilon(\tau) \qquad \varepsilon \subseteq \mathcal{E} = \{\bot, r, w, a\} \uplus \mathbb{E}$$

We write bool for $\mathsf{unit} + \mathsf{unit}$. Function types are restricted to be from values to computations as this is all we shall need to interpret a CBV source language. The effects which we detect are possible failure to terminate ($\bot$), $r$eading from a reference, $w$riting to a reference, $a$llocating a new reference cell and raising a particular exception $E \in \mathbb{E}$. Inclusion on sets of effects induces a subtyping relation:

$$\frac{}{\tau \leqslant \tau} \quad \tau \in \{\mathsf{unit}, \mathsf{int}, \mathsf{intref}\} \qquad \frac{\varepsilon \subseteq \varepsilon' \qquad \tau \leqslant \tau'}{\mathbf{T}_\varepsilon(\tau) \leqslant \mathbf{T}_{\varepsilon'}(\tau')}$$

$$\frac{\tau_1 \leqslant \tau_1' \qquad \tau_2 \leqslant \tau_2'}{\tau_1 \times \tau_2 \leqslant \tau_1' \times \tau_2'} \qquad \frac{\tau_1 \leqslant \tau_1' \qquad \tau_2 \leqslant \tau_2'}{\tau_1 + \tau_2 \leqslant \tau_1' + \tau_2'} \qquad \frac{\tau' \leqslant \tau \qquad \gamma \leqslant \gamma'}{\tau \to \gamma \leqslant \tau' \to \gamma'}$$

Reflexivity and transitivity are consequences of these rules.

There are two forms of typing judgment: $\Gamma \vdash V : \tau$ for values and $\Gamma \vdash M : \gamma$ for computations, where in both cases Γ is a finite map from term variables to *value* types (because the source language is CBV). We assume a countable set $\mathbb{L}$ of locations. The typing rules are shown in Figure 1 and satisfy the usual weakening, strengthening and substitution lemmas. We will sometimes use G to range over both value and computation terms and σ to range over both value and computation types. Most of the terms are unsurprising, but we do use a novel construct

$$\mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ \{E_1.M_1, \ldots, E_n.M_n\}\ \mathsf{in}\ N$$

which should be read "Evaluate the expression M. If successful, bind the result to x and evaluate N. Otherwise, if exception E_i is raised, evaluate the exception handler M_i instead, or if no handler is applicable, pass the exception on." There are many arguments in favour of this new construct, but particularly compelling for us is the fact that many optimising transformations simply cannot be expressed in terms of the more usual handle, and for this reason try is the construct actually used in the

$$\frac{}{\Gamma, x:\tau \vdash x:\tau} \qquad \frac{}{\Gamma \vdash \underline{n} : \mathsf{int}} \qquad \frac{}{\Gamma \vdash () : \mathsf{unit}} \qquad \frac{}{\Gamma \vdash \underline{\ell} : \mathsf{intref}} \ \ell \in \mathbb{L}$$

$$\frac{\Gamma \vdash V : \tau_i}{\Gamma \vdash \mathsf{in}_i V : \tau_1 + \tau_2} \ i = 1,2 \qquad \frac{\Gamma \vdash V_1 : \tau_1 \quad \Gamma \vdash V_2 : \tau_2}{\Gamma \vdash (V_1, V_2) : \tau_1 \times \tau_2}$$

$$\frac{\Gamma, x:\tau, f:\tau \to \mathbf{T}_{\varepsilon \cup \{\bot\}}(\tau') \vdash M : \mathbf{T}_{\varepsilon}(\tau')}{\Gamma \vdash (\mathsf{rec}\ f\ x = M) : \tau \to \mathbf{T}_{\varepsilon}(\tau')} \qquad \frac{\Gamma \vdash V : \tau_1}{\Gamma \vdash V : \tau_2} \ \tau_1 \leqslant \tau_2$$

$$\frac{\Gamma \vdash V_1 : \tau \to \gamma \quad \Gamma \vdash V_2 : \tau}{\Gamma \vdash V_1\ V_2 : \gamma} \qquad \frac{\Gamma \vdash V : \tau}{\Gamma \vdash \mathsf{val}\ V : \mathbf{T}_{\emptyset}(\tau)}$$

$$\frac{\Gamma \vdash M : \mathbf{T}_{\varepsilon}(\tau) \quad \Gamma \vdash H : \mathbf{T}_{\varepsilon'}(\tau') \quad \Gamma, x:\tau \vdash N : \mathbf{T}_{\varepsilon'}(\tau')}{\Gamma \vdash \mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ H\ \mathsf{in}\ N : \mathbf{T}_{\varepsilon \setminus \mathrm{dom}(H) \cup \varepsilon'}(\tau')} \qquad \frac{}{\Gamma \vdash \mathsf{raise}\ E : \mathbf{T}_{\{E\}}(\tau)}$$

$$\frac{\Gamma \vdash V : \tau_1 \times \tau_2}{\Gamma \vdash \pi_i V : \mathbf{T}_{\emptyset}(\tau_i)} \ i = 1,2 \qquad \frac{\Gamma \vdash V : \tau_1 + \tau_2 \quad \{\Gamma, x_i : \tau_i \vdash M_i : \gamma\}_{i=1,2}}{\Gamma \vdash (\mathsf{case}\ V\ \mathsf{of}\ \ \mathsf{in}_1 x_1.M_1\ ;\ \mathsf{in}_2 x_2.M_2) : \gamma}$$

$$\frac{\Gamma \vdash V : \mathsf{int}}{\Gamma \vdash \mathsf{ref}\ V : \mathbf{T}_{\{a\}}(\mathsf{intref})} \qquad \frac{\Gamma \vdash V : \mathsf{intref}}{\Gamma \vdash !V : \mathbf{T}_{\{r\}}(\mathsf{int})} \qquad \frac{\Gamma \vdash V_1 : \mathsf{intref} \quad \Gamma \vdash V_2 : \mathsf{int}}{\Gamma \vdash V_1 := V_2 : \mathbf{T}_{\{w\}}(\mathsf{unit})}$$

$$\frac{\Gamma \vdash V_1 : \mathsf{int} \quad \Gamma \vdash V_2 : \mathsf{int}}{\Gamma \vdash V_1 + V_2 : \mathbf{T}_{\emptyset}(\mathsf{int})} \qquad \frac{\Gamma \vdash V_1 : \mathsf{int} \quad \Gamma \vdash V_2 : \mathsf{int}}{\Gamma \vdash V_1 = V_2 : \mathbf{T}_{\emptyset}(\mathsf{bool})} \qquad \frac{\Gamma \vdash M : \gamma_1}{\Gamma \vdash M : \gamma_2} \ \gamma_1 \leqslant \gamma_2$$

Fig. 1. Typing rules for MIL-lite

MLj compiler. Also, we shall see that it permits a structurally-inductive definition of termination to be expressed directly in the syntax of terms. For now, observe that it nicely generalises both handle and Moggi's let, as illustrated by some of the syntactic sugar defined in Figure 2.

For ease of presentation the handlers are treated as a set in which no exception E appears more than once. We let H range over such sets, and write $H \setminus E$ to denote H with the handler for E removed (if it exists). We sometimes use map-like notation, for example writing $H(E)$ for the term M in a handler $E.M \in H$, and writing $\mathrm{dom}(H)$ for $\{E \mid E.M \in H\}$. We write $\Gamma \vdash H : \gamma$ to mean that for all $E.M \in H$, $\Gamma \vdash M : \gamma$.

2.2 The analysis

The way in which the MIL-lite typing rules express a simple effects analysis should be fairly clear, though some features may deserve further comment. The $\to$ introduction rule incorporates an extremely feeble, but nonetheless very useful, termination test: the more obvious rule would insist that $\bot \in \varepsilon$, but that would prevent $\lambda x.M$ from getting the natural derived typing rule and would cause undesirable non-termination effects to appear in, particularly, curried recursive functions.

$\lambda x.M$	$\stackrel{\text{def}}{=}$	$\mathsf{rec}\ f\ x = M \quad (f \notin FV(M))$
Ω	$\stackrel{\text{def}}{=}$	$(\mathsf{rec}\ f\ x = f\ x)\,()$
false	$\stackrel{\text{def}}{=}$	$\mathsf{in}_1()$
true	$\stackrel{\text{def}}{=}$	$\mathsf{in}_2()$
$\mathsf{if}\ V\ \mathsf{then}\ M_2\ \mathsf{else}\ M_1$	$\stackrel{\text{def}}{=}$	$\mathsf{case}\ V\ \mathsf{of}\ \mathsf{in}_1 x_1.M_1; \mathsf{in}_2 x_2.M_2 \quad (x_i \notin FV(M_i))$
$\mathsf{let}\ x \Leftarrow M\ \mathsf{in}\ N$	$\stackrel{\text{def}}{=}$	$\mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ \{\}\ \mathsf{in}\ N$
$\mathsf{let}\ x_1 \Leftarrow M_1; x_2 \Leftarrow M_2\ \mathsf{in}\ N$	$\stackrel{\text{def}}{=}$	$\mathsf{let}\ x_1 \Leftarrow M_1\ \mathsf{in}\ \mathsf{let}\ x_2 \Leftarrow M_2\ \mathsf{in}\ N$
$M; N$	$\stackrel{\text{def}}{=}$	$\mathsf{let}\ x \Leftarrow M\ \mathsf{in}\ N \quad (x \notin FV(N))$
$M\ \mathsf{handle}\ H$	$\stackrel{\text{def}}{=}$	$\mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ H\ \mathsf{in}\ \mathsf{val}\ x$
$\mathsf{set}\ \{\ell_1 \mapsto n_1, \ldots, \ell_k \mapsto n_k\}$	$\stackrel{\text{def}}{=}$	$\underline{\ell_1} := \underline{n_1}\ ;\ \ldots\ ;\ \underline{\ell_k} := \underline{n_k}\ ;\ \mathsf{val}\ ()$
$\mathsf{assert}\ (\ell, n)$	$\stackrel{\text{def}}{=}$	$\mathsf{let}\ v \Leftarrow !\underline{\ell}; b \Leftarrow (v = \underline{n})\ \mathsf{in}\ \mathsf{if}\ b\ \mathsf{then}\ \mathsf{val}\ ()\ \mathsf{else}\ \Omega$
$\mathsf{assert}\ \{\ell_1 \mapsto n_1, \ldots, \ell_k \mapsto n_k\}$	$\stackrel{\text{def}}{=}$	$\mathsf{assert}\ (\ell_1, n_1)\ ;\ \ldots\ ;\ \mathsf{assert}\ (\ell_k, n_k)\ ;\ \mathsf{val}\ ()$

Fig. 2. Syntactic sugar

The use of subtyping increases the accuracy of the analysis compared with one which just uses simple types or subeffecting. Subtyping prevents the bidirectional unification of effect information which would otherwise occur when several values flow to a common point (Peyton Jones and Wansbrough call this the 'poisoning problem' [7]).

There are many possible variants of the rules. For example, there is a stronger (try) rule in which the effects of the handlers are not all required to be the same, and only the effects of handlers corresponding to exceptions occurring in ε are unioned into the effect of the whole expression.

2.3 Operational semantics

We present the operational semantics of MIL-lite using a big-step evaluation relation $\Sigma, M \Downarrow \Sigma', R$ where R ranges over value terms and exception identifiers and $\Sigma \in \mathsf{States} \stackrel{\text{def}}{=} \mathbb{L} \rightharpoonup_{\text{fin}} \mathbb{Z}$. Write $\Sigma, M \Downarrow$ if $\Sigma, M \Downarrow \Sigma', R$ for some Σ', R and $\mathrm{locs}(G)$ for the set of location names occuring in G. If $\Sigma, \Delta \in \mathsf{States}$ then $(\Sigma \triangleleft \Delta) \in \mathsf{States}$ is defined by $(\Sigma \triangleleft \Delta)(\ell) = \Delta(\ell)$ if that's defined and $\Sigma(\ell)$ otherwise.

Lemma 2.1 (Type soundness) *If* $M : \mathbf{T}_\varepsilon(\tau)$ *and* $\mathrm{locs}(M) \subseteq \mathrm{dom}\,\Sigma$ *then if* $\Sigma, M \Downarrow \Sigma', V$ *then* $V : \tau$ *and* $\mathrm{locs}(V) \subseteq \mathrm{dom}\,\Sigma' \supseteq \mathrm{dom}\,\Sigma$. □

It is also easy to show that evaluation is unaffected by irrelevant locations and renaming:

Lemma 2.2 *If* $\mathrm{dom}\,\Sigma = \mathrm{locs}(M)$, $\mathrm{dom}\,\Sigma' \cap \mathrm{dom}\,\overline{\Sigma} = \emptyset$ *and* $\phi : \mathbb{L} \to \mathbb{L}$ *is a bijection then* $\Sigma, M \Downarrow \Sigma', R$ *if and only if* $(\Sigma \uplus \overline{\Sigma}) \circ \phi^{-1}, \phi(M) \Downarrow (\Sigma' \uplus \overline{\Sigma}) \circ \phi^{-1}, \phi(R)$ *with the obvious definition of the action of a renaming on expressions.* □

$$\Sigma, \mathsf{val}\ V \Downarrow \Sigma, V \qquad \Sigma, \mathsf{raise}\ E \Downarrow \Sigma, E \qquad \Sigma, \pi_i(V_1, V_2) \Downarrow \Sigma, V_i$$

$$\Sigma, \underline{n} + \underline{m} \Downarrow \Sigma, \underline{n+m} \qquad \Sigma, \underline{n} = \underline{n} \Downarrow \Sigma, \mathsf{true} \qquad \Sigma, \underline{n} = \underline{m} \Downarrow \Sigma, \mathsf{false}\ (n \neq m)$$

$$\Sigma, !\underline{\ell} \Downarrow \Sigma, \underline{\Sigma(\ell)} \qquad \Sigma, \underline{\ell} := \underline{n} \Downarrow \Sigma[\ell \mapsto n], () \qquad \Sigma, \mathsf{ref}\ \underline{n} \Downarrow \Sigma \uplus [\ell \mapsto n], \underline{\ell}$$

$$\frac{\Sigma, M_i[V/x_i] \Downarrow \Sigma', R}{\Sigma, \mathsf{case}\ \mathsf{in}_i V\ \mathsf{of}\ \mathsf{in}_1 x_1.M_1\ ;\ \mathsf{in}_2 x_2.M_2 \Downarrow \Sigma', R} \quad i = 1, 2$$

$$\frac{\Sigma, M[V/x, (\mathsf{rec}\ f\ x = M)/f] \Downarrow \Sigma', R}{\Sigma, (\mathsf{rec}\ f\ x = M)\ V \Downarrow \Sigma', R} \qquad \frac{\Sigma, M \Downarrow \Sigma', V \quad \Sigma', N[V/x] \Downarrow \Sigma'', R}{\Sigma, \mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ H\ \mathsf{in}\ N \Downarrow \Sigma'', R}$$

$$\frac{\Sigma, M \Downarrow \Sigma', E \quad \Sigma', M' \Downarrow \Sigma'', R}{\Sigma, \mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ H\ \mathsf{in}\ N \Downarrow \Sigma'', R} \quad H(E) = M'$$

$$\frac{\Sigma, M \Downarrow \Sigma', E}{\Sigma, \mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ H\ \mathsf{in}\ N \Downarrow \Sigma', E} \quad E \notin \mathrm{dom}(H)$$

Fig. 3. Evaluation relation for MIL-lite

2.4 Termination predicate

We now reformulate the operational semantics of MIL-lite in terms of a structurally defined termination predicate. This is essentially the same as the continuation-based termination relation used by Pitts and Stark [13,12] though our choice of unwieldy but well-behaved syntax allows us to express our version without introducing any extra syntax for continuation stacks. The reason for using this alternative presentation of the operational semantics is that it makes the proofs of the ciu theorem and the unwinding theorem more straightforward, by removing the need to deal explicitly with intermediate states and values. The termination predicate is defined over pairs of states and try terms by the rules shown in Figure 4. The relation between the two presentations of the operational semantics is then given by the following:

Lemma 2.3 *For all Σ, M, we have $\Sigma, M \Downarrow$ iff $\Sigma, \mathsf{let}\ x \Leftarrow M\ \mathsf{in}\ \mathsf{val}\ x \downarrow$.* □

2.5 Observational congruence and 'ciu' equivalence

In order to justify program transformations, we need a notion of when two expressions in our language are equivalent. The standard notion is that two terms are *contextually equivalent* if they may be textually interchanged in any program without affecting its observable behaviour. For MIL-lite, we take a program to be a closed term of any computation type and we observe whether or not its evaluation terminates. As usual, however, we find it convenient to work with a 'context-free' characterisation of contextual equivalence.

Definition 2.4 *Suppose R is a set of quadruples $(\Gamma, G_1, G_2, \sigma)$ such that whenever $(\Gamma, G_1, G_2, \sigma) \in R$ we have $\Gamma \vdash G_i : \sigma$ for $i \in \{1, 2\}$. Then we write $\Gamma \vdash G_1\ R\ G_2 : \sigma$ for $(\Gamma, G_1, G_2, \sigma) \in R$ and*

$$\frac{\Sigma, \mathsf{try}\ x \Leftarrow M_i[V/x_i]\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow (\mathsf{case}\ \mathsf{in}_i V\ \mathsf{of}\ \ \mathsf{in}_1 x_1.M_1\ ;\ \mathsf{in}_2 x_2.M_2)\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow}$$

$$\frac{\Sigma, \mathsf{try}\ x \Leftarrow M[V/y, (\mathsf{rec}\ f\ y = M)/f]\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow (\mathsf{rec}\ f\ y = M)V\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow} \qquad \frac{\Sigma, \mathsf{let}\ x \Leftarrow \mathsf{val}\ \underline{n_1 + n_2}\ \mathsf{in}\ N \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow \underline{n_1} + \underline{n_2}\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow}$$

$$\frac{\Sigma, \mathsf{let}\ x \Leftarrow \mathsf{val}\ V_i\ \mathsf{in}\ N \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow \pi_i(V_1, V_2)\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow} \qquad \frac{\Sigma[\ell \mapsto n], \mathsf{let}\ x \Leftarrow \mathsf{val}\ ()\ \mathsf{in}\ N \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow \underline{\ell} := \underline{n}\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow}$$

$$\frac{\Sigma, \mathsf{let}\ x \Leftarrow \mathsf{val}\ \underline{(\Sigma(\ell))}\ \mathsf{in}\ N \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow !\underline{\ell}\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow} \qquad \frac{\Sigma \uplus [\ell \mapsto n], \mathsf{let}\ x \Leftarrow \mathsf{val}\ \underline{\ell}\ \mathsf{in}\ N \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow \mathsf{ref}\ \underline{n}\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow}$$

$$\frac{\Sigma, \mathsf{let}\ x \Leftarrow \mathsf{val}\ \mathsf{true}\ \mathsf{in}\ N \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow \underline{n} = \underline{n}\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow} \qquad \frac{\Sigma, \mathsf{let}\ x \Leftarrow \mathsf{val}\ \mathsf{false}\ \mathsf{in}\ N \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow \underline{n} = \underline{m}\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow}\ n \neq m$$

$$\frac{}{\Sigma, \mathsf{try}\ x \Leftarrow \mathsf{val}\ V\ \mathsf{catch}\ H\ \mathsf{in}\ \mathsf{val}\ x \downarrow} \qquad \frac{}{\Sigma, \mathsf{try}\ x \Leftarrow \mathsf{raise}\ E\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow} E \notin \mathrm{dom}(H)$$

$$\frac{\Sigma, \mathsf{let}\ y \Leftarrow N[V/x]\ \mathsf{in}\ \mathsf{val}\ y \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow \mathsf{val}\ V\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow} N \neq \mathsf{val}\ x \qquad \frac{\Sigma, \mathsf{let}\ y \Leftarrow M\ \mathsf{in}\ \mathsf{val}\ y \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow \mathsf{raise}\ E\ \mathsf{catch}\ H\ \mathsf{in}\ N \downarrow} H(E) = M$$

$$\frac{\Sigma, \mathsf{try}\ y \Leftarrow M_1\ \mathsf{catch}\ (H_1\ \mathsf{catch}\ H_2)\ \mathsf{in}\ \mathsf{try}\ x \Leftarrow M_2\ \mathsf{catch}\ H_2\ \mathsf{in}\ N \downarrow}{\Sigma, \mathsf{try}\ x \Leftarrow (\mathsf{try}\ y \Leftarrow M_1\ \mathsf{catch}\ H_1\ \mathsf{in}\ M_2)\ \mathsf{catch}\ H_2\ \mathsf{in}\ N \downarrow}$$

where $H\ \mathsf{catch}\ H' \stackrel{\text{def}}{=} \{E.(M\ \mathsf{handle}\ H') \mid E.M \in H\} \cup (H' \setminus \mathrm{dom}(H))$

Fig. 4. Termination predicate

(i) *Write* $\widehat{R}$ *for the* compatible refinement *of* R *which is the typed relation which relates terms with the same outermost constructor and immediate subterms related by* R *[4]. Say* R *is* compatible *if it includes its compatible refinement. Note that compatibility implies reflexivity.*

(ii) R *is* closed under substitutions *if* $\vec{x_i} : \vec{\tau_i} \vdash G_1\ R\ G_2 : \sigma$ *implies that for all* $\vec{V_i} : \vec{\tau_i}$ *we have* $(G_1[V_i/x_i])\ R\ (G_2[V_i/x_i]) : \sigma$*. If* R *is a typed relation on closed terms, then* R° *is its* open extension*, the smallest relation containing* R *which is closed under substitutions.*

(iii) R *is* subsumptive *if it preserves the subtyping relation.*

(iv) R *is a* precongruence *if it is compatible, closed under substitutions, subsumptive and transitive. A* congruence *is a symmetric precongruence.*

(v) R *is* adequate *if for all closed computation terms* M_1*,* M_2 *with* $M_1\ R\ M_2 : \gamma$ *and for all* Σ *with* $\mathrm{locs}(M_1, M_2) \subseteq \mathrm{dom}\,\Sigma$*, if* $\Sigma, M_1 \Downarrow$ *then* $\Sigma, M_2 \Downarrow$*.*

Lemma 2.5 *There is a largest adequate congruence relation, which we call* observational congruence *and write* $\cong$*.* □

Lemma 2.6 *Observational congruence coincides with contextual equivalence.* □

We now characterise observational congruence using a Mason and Talcott style ciu ('closed instantiation of uses') equivalence [8] in a form due to Pitts [13]. As a proof principle, ciu equivalence is often less easy to apply than logical relations or bisimulation, but it suffices (together with the logical predicate which we will introduce later) to validate the fairly 'structural' rewrites in which we are interested.

Definition 2.7 *If* $M_1 : \mathbf{T}_\varepsilon(\tau)$ *and* $M_2 : \mathbf{T}_\varepsilon(\tau)$ *we write* $M_1 \leq_{ciu} M_2 : \mathbf{T}_\varepsilon(\tau)$ *if* $\forall N, H$ *such that* $x : \tau \vdash N : \gamma$ *and* $\vdash H : \gamma$, *and* $\forall \Sigma \in \mathit{States}$ *such that* $\mathrm{dom}\,\Sigma \supseteq \mathrm{locs}(M_1, M_2, H, N)$ *we have*

$$\Sigma, \mathsf{try}\, x \Leftarrow M_1 \,\mathsf{catch}\, H \,\mathsf{in}\, N \downarrow \quad \Longrightarrow \quad \Sigma, \mathsf{try}\, x \Leftarrow M_2 \,\mathsf{catch}\, H \,\mathsf{in}\, N \downarrow$$

If $V_1 : \tau$ *and* $V_2 : \tau$ *then we write* $V_1 \leq_{ciu} V_2 : \tau$ *for* $\mathsf{val}\, V_1 \leq_{ciu} \mathsf{val}\, V_2 : \mathbf{T}_\emptyset(\tau)$ *(cf. Moggi's mono condition). Write* $G_1 \cong_{ciu} G_2 : \sigma$ *and say* G_1 and G_2 are ciu equivalent at type σ *if* $G_1 \leq_{ciu} G_2 : \sigma$ *and* $G_2 \leq_{ciu} G_1 : \sigma$.

Lemma 2.8 *The relation* $\leq^\circ_{ciu}$ *is reflexive, transitive, closed under substitutions, subsumptive and adequate.* □

This isn't quite enough to let us use ciu equivalence to derive observational congruences, however. We also need to show that $\leq^\circ_{\mathsf{ciu}}$ is compatible (and hence a precongruence), for which we use a version of Howe's method [5]. We omit the details, which are in any case similar to those of other Howe's method proofs, but, briefly, we define a *precongruence candidate* relation $\preceq^\bullet$ by

$$\frac{\Gamma \vdash G \widehat{\preceq^\bullet} G'' : \sigma \quad \sigma \leqslant \sigma' \quad \Gamma \vdash G'' \leq^\circ_{\mathsf{ciu}} G' : \sigma'}{\Gamma \vdash G \preceq^\bullet G' : \sigma'}$$

and then prove a sequence of lemmas, of which the most important is the following, proved by termination induction:

Lemma 2.9 *If* $M_1 \preceq^\bullet M_2 : \mathbf{T}_\varepsilon(\tau)$ *and* $x : \tau \vdash N_1 \preceq^\bullet N_2 : \gamma$ *and* $H_1 \preceq^\bullet H_2 : \gamma$, *and* $\mathrm{dom}\,\Sigma \supseteq \mathrm{locs}(M_i, H_i, N_i)$ *then*

$$\Sigma, \mathsf{try}\, x \Leftarrow M_1 \,\mathsf{catch}\, H_1 \,\mathsf{in}\, N_1 \downarrow \quad \Longrightarrow \quad \Sigma, \mathsf{try}\, x \Leftarrow M_2 \,\mathsf{catch}\, H_2 \,\mathsf{in}\, N_2 \downarrow$$

□

We then deduce that the precongruence candidate relation coincides with ciu approximation and hence

Corollary 2.10 (ciu) *Ciu approximation coincides with observational precongruence and ciu equivalence coincides with observational congruence:*

$$\Gamma \vdash G \leq^\circ_{ciu} G' : \sigma \iff \Gamma \vdash G \leq G' : \sigma$$
$$\Gamma \vdash G \cong^\circ_{ciu} G' : \sigma \iff \Gamma \vdash G \cong G' : \sigma$$

□

2.6 The unwinding theorem

We will need a 'compactness of evaluation' result to establish the admissibility of the predicates which are introduced in the next section. To this end we inductively define the following sequence of finite approximations to recursive functions:

$$(\mathsf{rec}^0\ f\ x = M) \stackrel{\text{def}}{=} (\mathsf{rec}\ f\ x = f\ x)$$
$$(\mathsf{rec}^{n+1}\ f\ x = M) \stackrel{\text{def}}{=} (\mathsf{rec}\ f\ x = M[(\mathsf{rec}^n\ f\ x = M)/f])$$

It is easy to see that if $\Gamma \vdash \mathsf{rec}\ f\ x = M : \tau \to \gamma$ then for all $n \geq 1$, $\Gamma \vdash \mathsf{rec}^n\ f\ x = M : \tau \to \gamma$. This also holds for $n = 0$ iff the effect in γ contains $\bot$.

Lemma 2.11 *Let $P = \mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ H\ \mathsf{in}\ N$ with $g : \tau \to \mathbf{T}_{\varepsilon'}(\tau') \vdash P : \gamma$. If $\vdash (\mathsf{rec}\ f\ y = F) : \tau \to \mathbf{T}_{\varepsilon'}(\tau')$ and $\mathrm{dom}\,\Sigma \supseteq \mathrm{locs}(M, H, N, F)$ then*

(i) *If $\bot \in \varepsilon'$ then for all $G : \tau \to \mathbf{T}_{\varepsilon'}(\tau')$, if $\Sigma, P[(\mathsf{rec}^0\ f\ y = F)/g] \downarrow$ then $\Sigma, P[G/g] \downarrow$.*

(ii) *For all $n \in \mathbb{N}$, if $\Sigma, P[(\mathsf{rec}^n\ f\ y = F)/g] \downarrow$ then $\Sigma, P[(\mathsf{rec}^{n+1}\ f\ y = F)/g] \downarrow$.*

(iii) *If $\Sigma, P[(\mathsf{rec}\ f\ y = F)/g] \downarrow$ then there exists $n \in \mathbb{N}$ such that $\Sigma, P[(\mathsf{rec}^n\ f\ y = F)/g] \downarrow$.*

(iv) *For all $n \in \mathbb{N}$, If $\Sigma, P[(\mathsf{rec}^n\ f\ y = F)/g] \downarrow$ then $\Sigma, P[(\mathsf{rec}\ f\ y = F)/g] \downarrow$.*

Proof. By termination induction. □

Corollary 2.12 (Unwinding) *If $g : \tau \to \mathbf{T}_{\varepsilon'}(\tau') \vdash M : \mathbf{T}_{\varepsilon''}(\tau'')$, $\vdash (\mathsf{rec}\ f\ y = F) : \tau \to \mathbf{T}_{\varepsilon'}(\tau')$ and $\mathrm{dom}\,\Sigma \supseteq \mathrm{locs}(M, F)$ then $\Sigma, M[(\mathsf{rec}\ f\ y = F)/g] \Downarrow$ if and only if $\exists m \in \mathbb{N}. \forall n \geq m. \Sigma, M[(\mathsf{rec}^n\ f\ y = F)/g] \Downarrow$.* □

3 Semantics of Effects

In order to validate program equivalences which rely on effect information, we have to formalise what each of our effect-refined types actually means in terms of the operational semantics. For example, the meaning of a computation type $\mathbf{T}_\varepsilon(\tau)$ where $\bot \notin \varepsilon$ will be a set of terms whose evaluation in any state, amongst other things, does not diverge. We take the position that this meaning should not be tied too closely to the (in our case, rather weak) inference system used to *deduce* that a term has a particular property. In particular, the meaning of a property should be closed under observational equivalence in an appropriate sense. [1]

We shall express the intended meaning $[\![\sigma]\!]$ of each type σ in our language as the set of closed terms of that type which pass all of a collection of cotermination tests $\mathsf{Tests}_\sigma \subseteq \mathsf{States} \times \mathsf{Ctxt}_\sigma \times \mathsf{Ctxt}_\sigma$ where Ctxt_σ is the set of closed contexts with a finite number of holes of type σ. Formally:

$$[\![\sigma]\!] \stackrel{\text{def}}{=} \{\, G : \sigma \mid \forall (\Sigma, M[\cdot], M'[\cdot]) \in \mathsf{Tests}_\sigma.$$
$$\mathrm{locs}(M[G], M'[G]) \subseteq \mathrm{dom}\,\Sigma \implies (\Sigma, M[G] \Downarrow \iff \Sigma, M'[G] \Downarrow) \,\}$$

We define Tests_σ inductively as shown in Figure 5. Although these definitions appear

[1] This point is slightly obscured by the fact that we have combined the effect inference with a more conventional type system, rather than layering one over the other. The idea is that if two terms are observationally congruent at an unannotated type (equivalently, the type with the top effect annotation everywhere), then the predicate associated with a more refined annotation of that type should not distinguish them.

$$\mathsf{Tests}_{\mathsf{int}} \stackrel{\text{def}}{=} \{\} \qquad \mathsf{Tests}_{\mathsf{intref}} \stackrel{\text{def}}{=} \{\} \qquad \mathsf{Tests}_{\mathsf{unit}} \stackrel{\text{def}}{=} \{\}$$

$$\mathsf{Tests}_{\tau_1 \times \tau_2} \stackrel{\text{def}}{=} \bigcup_{i=1,2} \{(\Sigma, M[\pi_i[\cdot]], M'[\pi_i[\cdot]]) \mid (\Sigma, M[\cdot], M'[\cdot]) \in \mathsf{Tests}_{\tau_i}\}$$

$$\mathsf{Tests}_{\tau_1 + \tau_2} \stackrel{\text{def}}{=} \bigcup_{i=1,2} \{(\Sigma, \mathsf{case}\ [\cdot]\ \mathsf{of}\ \mathsf{in}_i x.M[x]\ ;\ \mathsf{in}_{3-i} y.\Omega,$$
$$\mathsf{case}\ [\cdot]\ \mathsf{of}\ \mathsf{in}_i x.M'[x]\ ;\ \mathsf{in}_{3-i} y.\Omega) \mid (\Sigma, M[\cdot], M'[\cdot]) \in \mathsf{Tests}_{\tau_i}\}$$

$$\mathsf{Tests}_{\tau \to \gamma} \stackrel{\text{def}}{=} \{(\Sigma, M[[\cdot]\ V], M'[[\cdot]\ V]) \mid V \in [\![\tau]\!],\ (\Sigma, M[\cdot], M'[\cdot]) \in \mathsf{Tests}_{\gamma}\}$$

$$\mathsf{Tests}_{\mathbf{T}_\varepsilon \tau} \stackrel{\text{def}}{=} \{(\Sigma, \mathsf{let}\ x \Leftarrow [\cdot]\ \mathsf{in}\ \mathsf{set}\ \Sigma'; M[x], \mathsf{let}\ x \Leftarrow [\cdot]\ \mathsf{in}\ \mathsf{set}\ \Sigma'; M'[x])$$
$$\mid (\Sigma', M[\cdot], M'[\cdot]) \in \mathsf{Tests}_{\tau},\ \Sigma \in \mathsf{States}\} \ \cup \bigcup_{e \notin \varepsilon} \mathsf{Tests}_{\overline{e},\tau}$$

where

$$\mathsf{Tests}_{\overline{\top},\tau} \stackrel{\text{def}}{=} \{(\Sigma, [\cdot], \mathsf{val}\ ()) \mid \Sigma \in \mathsf{States}\}$$

$$\mathsf{Tests}_{\overline{w},\tau} \stackrel{\text{def}}{=} \{(\Sigma, \mathsf{let}\ y \Leftarrow !\underline{\ell}\ \mathsf{in}\ \mathsf{try}\ x \Leftarrow [\cdot]\ \mathsf{catch}\ E.M\ \mathsf{in}\ N,$$
$$\mathsf{try}\ x \Leftarrow [\cdot]\ \mathsf{catch}\ E.\mathsf{let}\ y \Leftarrow !\underline{\ell}\ \mathsf{in}\ M\ \mathsf{in}\ \mathsf{let}\ y \Leftarrow !\underline{\ell}\ \mathsf{in}\ N)$$
$$\mid y : \mathsf{int}, x : \tau \vdash N : \gamma, y : \mathsf{int} \vdash M : \gamma, \Sigma \in \mathsf{States}, \ell \in \mathrm{dom}\,\Sigma\}$$

$$\mathsf{Tests}_{\overline{r},\tau} \stackrel{\text{def}}{=} \{(\Sigma, \mathsf{d}(\Sigma, \Delta, E); \mathsf{try}\ x \Leftarrow [\cdot]\ \mathsf{catch}\ E.\mathsf{assert}\ \Sigma \triangleleft \Delta; \mathsf{raise}\ E\ \mathsf{in}\ N,$$
$$\mathsf{d}(\Sigma, \Delta, E); \underline{\ell} := \underline{n}; \mathsf{try}\ x \Leftarrow [\cdot]\ \mathsf{catch}\ E.\mathsf{assert}\ \Sigma[\ell \mapsto n] \triangleleft \Delta; \mathsf{raise}\ E$$
$$\mathsf{in}\ \mathsf{assert}\ (\ell, (\Sigma[\ell \mapsto n] \triangleleft \Delta)(\ell)); \underline{\ell} := \underline{(\Sigma \triangleleft \Delta)(\ell)}; N)$$
$$\mid E \in \mathbb{E}, \Sigma, \Delta \in \mathsf{States}, \mathrm{dom}\,\Delta \subseteq \mathrm{dom}\,\Sigma \ni \ell, n \in \mathbb{Z}, x : \tau \vdash N : \gamma\}$$
$$\cup \{(\Sigma, [\cdot]\ \mathsf{handle}\ E.\Omega, \mathsf{set}\ \Sigma'; [\cdot]\ \mathsf{handle}\ E.\Omega) \mid \Sigma, \Sigma' \in \mathsf{States}, E \in \mathbb{E}\}$$

$$\mathsf{Tests}_{\overline{E},\tau} \stackrel{\text{def}}{=} \{(\Sigma, [\cdot], [\cdot]\ \mathsf{handle}\ E.N) \mid \Sigma \in \mathsf{States}, \vdash N : \gamma\}$$

$$\mathsf{Tests}_{\overline{a},\tau} \stackrel{\text{def}}{=} \{(\Sigma, \mathsf{let}\ x \Leftarrow [\cdot]; y \Leftarrow (\mathsf{set}\ \Sigma; [\cdot])\ \mathsf{in}\ N, \mathsf{let}\ x \Leftarrow [\cdot]; y \Leftarrow \mathsf{val}\ x\ \mathsf{in}\ N)$$
$$\mid \Sigma \in \mathsf{States}, x : \tau, y : \tau \vdash N : \gamma\}$$

and

$$\mathsf{K}_\Sigma n \stackrel{\text{def}}{=} \{\ell \mapsto n \mid \ell \in \mathrm{dom}(\Sigma)\}$$

$$\mathsf{d}(\Sigma, \Delta, E) \stackrel{\text{def}}{=} \mathsf{set}\ \mathsf{K}_\Sigma 0; (([\cdot]; \mathsf{val}\ ())\ \mathsf{handle}\ E.\mathsf{val}\ ()); \mathsf{assert}\ \mathsf{K}_\Sigma 0 \triangleleft \Delta;$$
$$\mathsf{set}\ \mathsf{K}_\Sigma 1; (([\cdot]; \mathsf{val}\ ())\ \mathsf{handle}\ E.\mathsf{val}\ ()); \mathsf{assert}\ \mathsf{K}_\Sigma 1 \triangleleft \Delta; \mathsf{set}\ \Sigma$$

Fig. 5. Definition of Tests_σ

rather complex, at value types they actually amount to a familiar-looking logical predicate:

Lemma 3.1

- $[\![\mathsf{int}]\!] = \{\underline{n} \mid n \in \mathbb{Z}\}$, $[\![\mathsf{intref}]\!] = \{\underline{\ell} \mid \ell \in \mathbb{L}\}$ *and* $[\![\mathsf{unit}]\!] = \{()\}$.
- $[\![\tau_1 \times \tau_2]\!] = \{(V_1, V_2) \mid V_1 \in [\![\tau_1]\!], V_2 \in [\![\tau_2]\!]\}$
- $[\![\tau \to \gamma]\!] = \{F : \tau \to \gamma \mid \forall V \in [\![\tau]\!].(F\ V) \in [\![\gamma]\!]\}$
- $[\![\tau_1 + \tau_2]\!] = \bigcup_{i=1,2}\{\mathsf{in}_i V \mid V \in [\![\tau_i]\!]\}$
- $M : \mathbf{T}_\varepsilon(\tau) \in [\![\mathbf{T}_\varepsilon(\tau)]\!]$ *iff for any* Σ *such that* $\mathrm{locs}(M) \subseteq \mathrm{dom}(\Sigma)$,
  - *if* $\Sigma, M \Downarrow \Sigma', V$ *then* $V \in [\![\tau]\!]$.
  - *if* $\bot \notin \varepsilon$ *then* $\Sigma, M \Downarrow \Sigma', R$ *for some state* Σ' *and result* R.
  - *if* $E \notin \varepsilon$ *and* $\Sigma, M \Downarrow \Sigma', E'$ *then* $E \neq E'$.
  - *if* $w \notin \varepsilon$ *and* $\Sigma, M \Downarrow \Sigma', R$ *then* $\Sigma' = \Sigma \uplus \overline{\Sigma}$ *for some state* $\overline{\Sigma}$.
  - *if* $a \notin \varepsilon$ *and* $\Sigma, M \Downarrow \Sigma' \uplus \overline{\Sigma}, V$ *for* $\mathrm{dom}(\Sigma) = \mathrm{dom}(\Sigma')$ *then for any term* N *and bijection* $\phi : \mathrm{dom}\,\overline{\Sigma} \to L$ *such that* $x : \tau, y : \tau \vdash N : \gamma$ *and* $\mathrm{locs}(N) \subseteq \mathrm{dom}(\Sigma)$ *and* $L \cap (\mathrm{dom}(\Sigma) \cup \mathrm{dom}(\overline{\Sigma})) = \emptyset$,

$$\Sigma' \uplus \overline{\Sigma}, N[V/x, V/y] \Downarrow \ \textit{iff}\ \ \Sigma' \uplus \overline{\Sigma} \uplus (\overline{\Sigma} \circ \phi^{-1}), N[V/x, \phi(V)/y] \Downarrow .$$

  - *if* $r \notin \varepsilon$ *then there exists some state change* Δ *with* $\mathrm{dom}(\Delta) \subseteq \mathrm{dom}(\Sigma)$ *such that for all* Σ' *with* $\mathrm{dom}(\Sigma') = \mathrm{dom}(\Sigma)$,

(i) *if* $\Sigma, M \Downarrow \Sigma \triangleleft \Delta \uplus \overline{\Sigma}, V$ *for some* $\overline{\Sigma}$ *and* V *then* $\Sigma', M \Downarrow \Sigma' \triangleleft \Delta \uplus \overline{\Sigma}', V'$ *for some* $\overline{\Sigma}'$ *and* V' *such that for any* N *with* $x : \tau \vdash N : \gamma$ *and* $\mathrm{locs}(N) \subseteq \mathrm{dom}(\Sigma)$,

$$\Sigma \triangleleft \Delta \uplus \overline{\Sigma}, N[V/x] \Downarrow \ \textit{iff}\ \ \Sigma \triangleleft \Delta \uplus \overline{\Sigma}', N[V'/x] \Downarrow .$$

(ii) *if* $\Sigma, M \Downarrow \Sigma \triangleleft \Delta \uplus \overline{\Sigma}, E$ *for some* $\overline{\Sigma}, E$ *then* $\Sigma', M \Downarrow \Sigma' \triangleleft \Delta \uplus \overline{\Sigma}', E$ *for some* $\overline{\Sigma}'$.

□

Lemma 3.2 *If* $\sigma \leqslant \sigma'$ *then* $[\![\sigma]\!] \subseteq [\![\sigma']\!]$. □

The advantage of the formulation in terms of tests is that it immediately allows us to use the Unwinding Theorem (Corollary 2.12) to deduce the following:

Lemma 3.3 (Admissibility) *If* $g : \tau \to \gamma \vdash G : \sigma$ *and* $\vdash \mathsf{rec}\, f\, x = N : \tau \to \gamma$ *then* $\exists m \in \mathbb{N}.\forall n \geq m.\ G[(\mathsf{rec}^n\ f\ x = N)/g] \in [\![\sigma]\!]$ *implies* $G[(\mathsf{rec}\, f\ x = N)/g] \in [\![\sigma]\!]$.

Proof. Assume $(\Sigma, M[\cdot], M'[\cdot]) \in \mathsf{Tests}_\sigma$ with $\mathrm{dom}\,\Sigma \supseteq \mathrm{locs}(G, N, M, M')$. Then

$$\begin{aligned}
&\Sigma, M[G[(\mathsf{rec}\ f\ x = N)/g]] \Downarrow \\
&\quad\iff \exists n \geq m.\ \Sigma, M[G[(\mathsf{rec}^n\ f\ x = N)/g]] \Downarrow \quad \text{Unwinding} \\
&\quad\iff \exists n \geq m.\ \Sigma, M'[G[(\mathsf{rec}^n\ f\ x = N)/g]] \Downarrow \quad \text{assumption} \\
&\quad\iff \Sigma, M'[G[(\mathsf{rec}\ f\ x = N)/g]] \Downarrow \quad \text{Unwinding}
\end{aligned}$$

□

This is essentially just the same argument as Pitts uses to show that his "$(\cdot)^{\top\top}$-closed relations" are admissible [12]. We are now in a position to prove the 'Fundamental Theorem' for our logical predicate:

Theorem 3.4 *If* $\vec{x_i} : \vec{\tau_i} \vdash G : \sigma$ *and* $\vec{V_i} \in [\![\vec{\tau_i}]\!]$ *then* $G[\vec{V_i}/\vec{x_i}] \in [\![\sigma]\!]$.

Proof. Induction on the derivation of $\vec{x_i} : \vec{\tau_i} \vdash G : \sigma$, using Lemma 3.3 in the case of recursive function definitions, Lemma 3.2 for the subtyping rule and more specific but straightforward reasoning for each of the effect-annotated computation types.□

Although we have explained the meaning of our logical predicate at value types, it seems worth commenting a little further on the definitions of $\mathsf{Tests}_{\overline{e},\tau}$. The intention is that the extent of $\mathsf{Tests}_{\overline{e},\tau}$ is the set of computations of type $\mathbf{T}_{\mathcal{E}}(\tau)$ which definitely do *not* have effect e. So, passing all the tests in $\mathsf{Tests}_{\overline{\perp},\tau}$ is easily seen to be equivalent to not diverging in any state and passing all the tests in $\mathsf{Tests}_{\overline{E},\tau}$ means not throwing exception E in any state.

The tests concerning store effects are a little more subtle. It is not too hard to see that $\mathsf{Tests}_{\overline{w},\tau}$ expresses not *observably* writing the store, but note that this is not the same thing as the more naive intensional property of never performing assignment during evaluation; a computation which, for example, incremented and then decremented a reference cell would pass our tests (though our *analysis* is too weak to detect such a property). Similarly, $\mathsf{Tests}_{\overline{r},\tau}$ tests (contortedly!) for not observably reading the store, by running the computation in different initial states and seeing if the results can be distinguished by a subsequent continuation.

The most surprising definition is probably that of $\mathsf{Tests}_{\overline{a},\tau}$, the extent of which is intended to be those computations which do not observably allocate any new storage locations. This should include, for example, a computation which allocates a reference and then returns a function which uses that reference to keep count of how many times it has been called, but which never reveals the counter, nor returns different results according to its value. However, the definition of $\mathsf{Tests}_{\overline{a},\tau}$ does not seem to say anything about store extension; what it actually captures is those computations for which two evaluations in equivalent initial states yield indistinguishable results. Our choice of this as the meaning of 'doesn't allocate' was guided by the optimising transformations which we wished to be able to perform rather than a deep understanding of exactly what it means to not allocate observably, but in retrospect it seems quite reasonable.

4 Equivalences

4.1 Effect-independent equivalences

Figure 6 presents some typed observational congruences that correspond to identities from the equational theory of the computational lambda calculus, and Figure 7 presents equivalences that involve local side-effecting behaviour. Directed variants of many of these are useful transformations that are in fact performed by MLj (although the duplication of terms in cc_2 is avoided by introducing an abstraction). These equations can be derived without recourse to our logical predicate, by making use of a rather strong notion of equivalence that can easily be shown to be contained in ciu equivalence. Note that our definition is more generous than corre-

$$\beta\text{-}\times\frac{\Gamma \vdash V_1 : \tau_1 \qquad \Gamma \vdash V_2 : \tau_2}{\Gamma \vdash \pi_i(V_1, V_2) \cong \mathsf{val}\, V_i : \mathbf{T}_\emptyset(\tau_i)} \qquad \beta\text{-}\mathbf{T}\frac{\Gamma \vdash V : \tau \qquad \Gamma, x : \tau \vdash M : \gamma}{\Gamma \vdash \mathsf{let}\, x \Leftarrow \mathsf{val}\, V\, \mathsf{in}\, M \cong M[V/x] : \gamma}$$

$$\beta\text{-}\to\frac{\Gamma, x : \tau, f : \tau \to \mathbf{T}_{\varepsilon\cup\{\bot\}}(\tau') \vdash M : \mathbf{T}_\varepsilon(\tau') \qquad \Gamma \vdash V : \tau}{\Gamma \vdash (\mathsf{rec}\, f\, x = M)\, V \cong M[V/x, \mathsf{rec}\, f\, x = M/f] : \mathbf{T}_\varepsilon(\tau')}$$

$$\beta\text{-}+\frac{\Gamma \vdash V : \tau_i \qquad \Gamma, x_1 : \tau_1 \vdash M_1 : \gamma \qquad \Gamma, x_2 : \tau_2 \vdash M_2 : \gamma}{\Gamma \vdash \mathsf{case}\, \mathsf{in}_i V\, \mathsf{of}\, \mathsf{in}_1 x_1.M_1; \mathsf{in}_2 x_2.M_2 \cong M_i[V/x_i] : \gamma}$$

$$\eta\text{-}\times\frac{\Gamma \vdash V : \tau_1 \times \tau_2}{\Gamma \vdash \mathsf{let}\, x_1 \Leftarrow \pi_1 V; x_2 \Leftarrow \pi_2 V\, \mathsf{in}\, \mathsf{val}\, (x_1, x_2) \cong \mathsf{val}\, V : \mathbf{T}_\emptyset(\tau_1 \times \tau_2)}$$

$$\eta\text{-}+\frac{\Gamma \vdash V : \tau_1 + \tau_2}{\Gamma \vdash \mathsf{case}\, V\, \mathsf{of}\, \mathsf{in}_1 x_1.\mathsf{val}\, (\mathsf{in}_1 x_1); \mathsf{in}_2 x_2.\mathsf{val}\, (\mathsf{in}_2 x_2) \cong \mathsf{val}\, V : \mathbf{T}_\emptyset(\tau_1 + \tau_2)}$$

$$\eta\text{-}\to\frac{\Gamma \vdash V : \tau \to \gamma}{\Gamma \vdash \mathsf{rec}\, f\, x = V\, x \cong V : \tau \to \gamma} \qquad \eta\text{-}\mathbf{T}\frac{\Gamma \vdash M : \gamma}{\Gamma \vdash \mathsf{let}\, x \Leftarrow M\, \mathsf{in}\, \mathsf{val}\, x \cong M : \gamma}$$

$$cc_1\frac{\Gamma \vdash M_1 : \mathbf{T}_{\varepsilon_1}(\tau_1) \qquad \Gamma, y : \tau_1 \vdash M_2 : \mathbf{T}_{\varepsilon_2}(\tau_2) \qquad \Gamma, y : \tau_1, x : \tau_2 \vdash M_3 : \mathbf{T}_{\varepsilon_3}(\tau_3)}{\Gamma \vdash \mathsf{let}\, x \Leftarrow (\mathsf{let}\, y \Leftarrow M_1\, \mathsf{in}\, M_2)\, \mathsf{in}\, M_3 \cong \mathsf{let}\, y \Leftarrow M_1; x \Leftarrow M_2\, \mathsf{in}\, M_3 : \mathbf{T}_{\varepsilon_1\cup\varepsilon_2\cup\varepsilon_3}(\tau_3)}$$

$$cc_2\frac{\Gamma \vdash V : \tau_1 + \tau_2 \quad \{\Gamma, x_i : \tau_i \vdash M_i : \mathbf{T}_\varepsilon(\tau)\} \quad \Gamma, x : \tau \vdash N : \mathbf{T}_{\varepsilon'}(\tau')}{\Gamma \vdash \mathsf{let}\, x \Leftarrow \mathsf{case}\, V\, \mathsf{of}\, \{\mathsf{in}_i x_i.M_i\}\, \mathsf{in}\, N \cong \mathsf{case}\, V\, \mathsf{of}\, \{\mathsf{in}_i x_i.\mathsf{let}\, x \Leftarrow M_i\, \mathsf{in}\, N\} : \mathbf{T}_{\varepsilon\cup\varepsilon'}(\tau')}$$

$$\beta\text{-}E\frac{\Gamma \vdash M : \gamma \qquad \Gamma \vdash H : \gamma \qquad \Gamma, x : \tau \vdash N : \gamma}{\Gamma \vdash \mathsf{try}\, x \Leftarrow \mathsf{raise}\, E\, \mathsf{catch}\, (E.M); H\, \mathsf{in}\, N \cong M : \gamma}$$

$$\eta\text{-}E\frac{\Gamma \vdash M : \mathbf{T}_\varepsilon(\tau) \qquad \Gamma \vdash H : \mathbf{T}_{\varepsilon'}(\tau') \qquad \Gamma, x : \tau \vdash N : \mathbf{T}_{\varepsilon'}(\tau')}{\Gamma \vdash \mathsf{try}\, x \Leftarrow M\, \mathsf{catch}\, (E.\mathsf{raise}\, E); H\, \mathsf{in}\, N \cong \mathsf{try}\, x \Leftarrow M\, \mathsf{catch}\, H\, \mathsf{in}\, N : \mathbf{T}_{\varepsilon\cup\varepsilon'}(\tau')}$$

Fig. 6. Effect-independent equivalences (1)

sponding definitions for similar calculi [11], permitting the final states to differ in their 'inaccessible' parts (*c.f.* Mason and Talcott's 'strong isomorphism' [8]).

Definition 4.1 (Kleene equivalence) *For two closed terms $M_1, M_2 : \gamma$ we write $M_1 \leq_{kl} M_2$ if for any $\Sigma, \Sigma', \overline{\Sigma}_1$ such that* $\mathrm{locs}(M_1, M_2) \subseteq \mathrm{dom}(\Sigma) \subseteq \mathrm{dom}(\Sigma') \supseteq \mathrm{locs}(R)$,

$$\Sigma, M_1 \Downarrow \Sigma' \uplus \overline{\Sigma}_1, R \quad \Rightarrow \quad \Sigma, M_2 \Downarrow \Sigma' \uplus \overline{\Sigma}_2, R \text{ for some } \overline{\Sigma}_2.$$

We write $M_1 \cong_{kl} M_2$ if $M_1 \leq_{kl} M_2$ and $M_2 \leq_{kl} M_1$ and say that M_1 and M_2 are Kleene equivalent.

Lemma 4.2 $\leq_{kl} \subseteq \leq_{ciu}$. □

The beta-equivalences and commuting conversions of Figure 6 together with the equivalences of Figure 7 are derived directly as Kleene equivalences. Derivation of the eta-equivalences involves first deriving a number of extensionality properties using ciu equivalence; similar techniques are used in [10].

$$\frac{\Gamma \vdash V : \mathsf{int} \qquad \Gamma \vdash M : \mathbf{T}_{\varepsilon}(\tau)}{\Gamma \vdash \mathsf{let}\ x \Leftarrow \mathsf{ref}\ V\ \mathsf{in}\ M \cong M : \mathbf{T}_{\varepsilon \cup \{a\}}(\tau)}$$

$$\frac{\Gamma \vdash V : \mathsf{intref} \qquad \Gamma, x : \mathsf{int}, y : \mathsf{int} \vdash M : \mathbf{T}_{\varepsilon}(\tau)}{\Gamma \vdash \mathsf{let}\ x \Leftarrow !V; y \Leftarrow !V\ \mathsf{in}\ M \cong \mathsf{let}\ x \Leftarrow !V; y \Leftarrow \mathsf{val}\ x\ \mathsf{in}\ M : \mathbf{T}_{\varepsilon \cup \{r\}}(\tau)}$$

$$\frac{\Gamma \vdash V_1 : \mathsf{int} \qquad \Gamma \vdash V_2 : \mathsf{int} \qquad \Gamma, x_1 : \mathsf{intref}, x_2 : \mathsf{intref} \vdash M : \mathbf{T}_{\varepsilon}(\tau)}{\Gamma \vdash \mathsf{let}\ x_1 \Leftarrow \mathsf{ref}\ V_1; x_2 \Leftarrow \mathsf{ref}\ V_2\ \mathsf{in}\ M \cong \mathsf{let}\ x_2 \Leftarrow \mathsf{ref}\ V_2; x_1 \Leftarrow \mathsf{ref}\ V_1\ \mathsf{in}\ M : \mathbf{T}_{\varepsilon \cup \{a\}}(\tau)}$$

$$\frac{\Gamma \vdash V_1 : \mathsf{intref} \qquad \Gamma \vdash V_2 : \mathsf{int} \qquad \Gamma, x : \mathsf{int} \vdash M : \mathbf{T}_{\varepsilon}(\tau)}{\Gamma \vdash V_1 := V_2; \mathsf{let}\ x \Leftarrow !V_1\ \mathsf{in}\ M \cong V_1 := V_2; M[V_2/x] : \mathbf{T}_{\varepsilon \cup \{r,w\}}(\tau)}$$

Fig. 7. Effect-independent equivalences (2)

$$\textit{discard}\frac{\Gamma \vdash M : \mathbf{T}_{\varepsilon_1}(\tau_1) \qquad \Gamma \vdash N : \mathbf{T}_{\varepsilon_2}(\tau_2)}{\Gamma \vdash \mathsf{let}\ x \Leftarrow M\ \mathsf{in}\ N \cong N : \mathbf{T}_{\varepsilon_1 \cup \varepsilon_2}\tau_2}$$

where $\varepsilon_1 \subseteq \{r, a\}$

$$\textit{copy}\frac{\Gamma \vdash M : \mathbf{T}_{\varepsilon}(\tau) \qquad \Gamma, x : \tau, y : \tau \vdash N : \mathbf{T}_{\varepsilon'}(\tau')}{\Gamma \vdash \mathsf{let}\ x \Leftarrow M; y \Leftarrow M\ \mathsf{in}\ N \cong \mathsf{let}\ x \Leftarrow M; y \Leftarrow \mathsf{val}\ x\ \mathsf{in}\ N : \mathbf{T}_{\varepsilon \cup \varepsilon'}(\tau')}$$

where $\{r, a\} \cap \varepsilon = \emptyset$ or $\{w, a\} \cap \varepsilon = \emptyset$

$$\textit{swap}\frac{\Gamma \vdash M_1 : \mathbf{T}_{\varepsilon_1}(\tau_1) \qquad \Gamma \vdash M_2 : \mathbf{T}_{\varepsilon_2}(\tau_2) \qquad \Gamma, x_1 : \tau_1, x_2 : \tau_2 \vdash N : \mathbf{T}_{\varepsilon_3}(\tau_3)}{\Gamma \vdash \mathsf{let}\ x_1 \Leftarrow M_1; x_2 \Leftarrow M_2\ \mathsf{in}\ N \cong \mathsf{let}\ x_2 \Leftarrow M_2; x_1 \Leftarrow M_1\ \mathsf{in}\ N : \mathbf{T}_{\varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3}(\tau_3)}$$

where $\varepsilon_1, \varepsilon_2 \subseteq \{r, a, \bot\}$ or $\varepsilon_1 \subseteq \{a, \bot\}, \varepsilon_2 \subseteq \{r, w, a, \bot\}$

$$\textit{dead-try}\frac{\Gamma \vdash M : \mathbf{T}_{\varepsilon}(\tau) \qquad \Gamma \vdash H : \mathbf{T}_{\varepsilon'}(\tau') \qquad \Gamma, x : \tau \vdash N : \mathbf{T}_{\varepsilon'}(\tau')}{\Gamma \vdash \mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ H\ \mathsf{in}\ N \cong \mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ H \setminus E\ \mathsf{in}\ N : \mathbf{T}_{\varepsilon \cup \varepsilon'}(\tau')}$$

where $E \notin \varepsilon$

Fig. 8. Effect-dependent equivalences

4.2 Effect-dependent equivalences

We now come to a set of equivalences that are dependent on effect information, which are shown in Figure 8. Notice how the first three of these equations respectively subsume the first three local equivalences of Figure 7. We will give the proofs for two representative cases.

Proposition 4.3 *The discard equivalence is valid.*

Proof. We show that the terms are Kleene-equivalent and thus observationally congruent by Lemma 4.2 and ciu equivalence.

It suffices to consider closed terms M and N. By Theorem 3.4 and Lemma 3.1 (cases E, w and $\bot$) we know for any state Σ with $\mathrm{locs}(M,N) \subseteq \mathrm{dom}(\Sigma)$ that $\Sigma, M \Downarrow \Sigma \uplus \overline{\Sigma}, V$ for some $V, \overline{\Sigma}$. Consider the evaluation of N.

(i) $\Sigma, N \Downarrow \Sigma', R$ for some Σ' and R. By Lemma 2.2 we know that $\Sigma \uplus \overline{\Sigma}, N \Downarrow \Sigma' \uplus \overline{\Sigma}, R$. Then by an application of the first rule for try in Figure 3 we obtain that $\Sigma, \mathsf{let}\ x \Leftarrow M\ \mathsf{in}\ N \Downarrow \Sigma' \uplus \overline{\Sigma}, R$.

(ii) $\Sigma, N \Uparrow$. By Lemma 2.2 we know $\Sigma \uplus \overline{\Sigma}, N \Uparrow$ and from the evaluation rules we have $\Sigma, \mathsf{let}\ x \Leftarrow M\ \mathsf{in}\ N \Uparrow$.

□

Proposition 4.4 *The copy equivalence is valid.*

Proof. It suffices to consider empty Γ. By Theorem 3.4 we know that $M \in [\![\mathbf{T}_\varepsilon(\tau)]\!]$. By the ciu theorem we must show for any Σ, P, H with $\mathrm{locs}(M,N,P,H) \subseteq \mathrm{dom}(\Sigma)$ and $\vdash H : \gamma$ and $z : \tau' \vdash P : \gamma$ that

$$\Sigma, \mathsf{try}\ z \Leftarrow (\mathsf{let}\ x \Leftarrow M; y \Leftarrow M\ \mathsf{in}\ N)\ \mathsf{catch}\ H\ \mathsf{in}\ P \Downarrow$$
$$\text{iff } \Sigma, \mathsf{try}\ z \Leftarrow (\mathsf{let}\ x \Leftarrow M; y \Leftarrow \mathsf{val}\ x\ \mathsf{in}\ N)\ \mathsf{catch}\ H\ \mathsf{in}\ P \Downarrow .$$

It is easy to see from the evaluation relation that it suffices to prove

$$\begin{aligned} &\Sigma, \mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ H\ \mathsf{in}\ \mathsf{try}\ y \Leftarrow M\ \mathsf{catch}\ H\ \mathsf{in}\ Q \Downarrow \\ &\text{iff } \Sigma, \mathsf{try}\ x \Leftarrow M\ \mathsf{catch}\ H\ \mathsf{in}\ \mathsf{let}\ y \Leftarrow \mathsf{val}\ x\ \mathsf{in}\ Q \Downarrow \end{aligned} \tag{1}$$

where $Q \stackrel{\mathrm{def}}{=} \mathsf{try}\ z \Leftarrow N\ \mathsf{catch}\ H\ \mathsf{in}\ P$. Clearly this holds if $\Sigma, M \Uparrow$ or if $\Sigma, M \Downarrow \Sigma', E$ for some Σ' and E. Therefore suppose that $\Sigma, M \Downarrow \Sigma \triangleleft \Delta \uplus \overline{\Sigma}, V$ for some $\Delta, \overline{\Sigma}, V$. Then by applying the appropriate evaluation rule (1) can be reduced to

$$\begin{aligned} &\Sigma \triangleleft \Delta \uplus \overline{\Sigma}, \mathsf{try}\ y \Leftarrow M\ \mathsf{catch}\ H\ \mathsf{in}\ Q[V/x] \Downarrow \\ &\text{iff } \Sigma \triangleleft \Delta \uplus \overline{\Sigma}, Q[V/x, V/y] \Downarrow . \end{aligned} \tag{2}$$

We consider two cases:

- $\{w, a\} \cap \varepsilon = \emptyset$. By Lemma 3.1 (case w) we know that $\Sigma \triangleleft \Delta = \Sigma$ and it follows from Lemma 2.2 that $\Sigma \uplus \overline{\Sigma}, M \Downarrow \Sigma \uplus \overline{\Sigma} \uplus \overline{\Sigma} \circ \phi^{-1}, \phi(V)$ for some bijection on locations ϕ. By applying the appropriate rule from the evaluation relation we can reduce the left-hand-side of (2) to

  $$\Sigma \uplus \overline{\Sigma} \uplus \overline{\Sigma} \circ \phi^{-1}, Q[V/x, \phi(V)/y] \Downarrow .$$

  Applying Lemma 3.1 (case a) gives the right-hand-side of (2) as required.

- $\{r, a\} \cap \varepsilon = \emptyset$. By Lemma 2.2 we know that $\Sigma \uplus \overline{\Sigma} \circ \phi, M \Downarrow \Sigma \triangleleft \Delta \uplus \overline{\Sigma} \circ \phi \uplus \overline{\Sigma}, V$ for an appropriate bijection ϕ. Then it follows from Lemma 3.1 (case r, letting $\Sigma' = \Sigma \triangleleft \Delta \uplus \overline{\Sigma} \circ \phi$ and $N = Q[\phi^{-1}(V)/x]$) that

  $$\Sigma \triangleleft \Delta \uplus \overline{\Sigma} \circ \phi, M \Downarrow \Sigma \triangleleft \Delta \uplus \overline{\Sigma} \circ \phi \uplus \overline{\Sigma}', V' \tag{3}$$

  for some $\overline{\Sigma}'$ and V', where without loss of generality we assume $\mathrm{dom}(\overline{\Sigma})$ to be disjoint from $\mathrm{dom}(\overline{\Sigma}')$, and that

$$\begin{aligned} &\Sigma \lhd \Delta \uplus \overline{\Sigma} \circ \phi \uplus \overline{\Sigma}, Q[\phi^{-1}(V)/x, V/y] \Downarrow \\ \text{iff } &\Sigma \lhd \Delta \uplus \overline{\Sigma} \circ \phi \uplus \overline{\Sigma'}, Q[\phi^{-1}(V)/x, V'/y] \Downarrow . \end{aligned} \tag{4}$$

By Lemma 3.1 (case a) the left-hand-side of (4) is equivalent to

$$\Sigma \lhd \Delta \uplus \overline{\Sigma}, Q[V/x, V/y] \Downarrow$$

and it follows from Lemma 2.2 that the right-hand-side is equivalent to

$$\Sigma \lhd \Delta \uplus \overline{\Sigma} \uplus \overline{\Sigma'}, Q[V/x, V'/y] \Downarrow .$$

Applying the appropriate evaluation rule gives the result as required.

□

5 Conclusions

The effect inference built into our type system is fairly weak, though compared with those of Wadler [16] and Tolmach [15], it deals with a richer language of effects and incorporates a more general notion of subtyping. We haven't discussed the inference algorithm here, but we use a fairly standard constraint-based algorithm for dealing with subtyping. A more subtle issue is how to maintain and refine effect information during rewriting: a rewritten term can often be re-typed with smaller effects, thus enabling further rewrites. It would be interesting to formalise rewrite rules on derivations (equivalently, term with explicit coercions) which 'push coercions around the program'.

The rewrites in this paper are all intuitively 'obvious', and our proofs of soundness are surprisingly involved. It is probably the case that we could have proved these exact results slightly more easily by, for example, using an instrumented operational semantics, particularly as our the analysis presented here is so intensional. The techniques used here should, however, scale up to a more precise analysis, such as one involving regions or logical combinations of properties. Another justification for working with observationally closed predicates is that there are a small number of places in MLj's implementation of the Standard Basis Library where we explicitly annotate expressions with a smaller effect than could be inferred by our type system; this allows, for example, imperative initialisation of various lookup tables to be dead-coded from programs which make no use of them. Verifying the soundness of these annotations in the present framework is much more straightforward than it would be had we used more intensional predicates.

The use of sets of contextual tests to express the meaning of types is interesting and worth further study. One alternative would be to define an operational logical relation closer to that used by Pitts and Stark [13], rather than a predicate. The fundamental theorem for such a relation should give the ciu theorem as a corollary, rather than requiring it to be proved separately.

There are a number of possible extensions to the underlying type system. We would expect that polymorphism and inductive types could be treated using essentially the same techniques, though mixed variance recursive types and higher type references are both likely to be more difficult.

We thank Gavin Bierman, Andy Gordon, Soren Lassen, Andy Pitts, George Russell and Ian Stark for many useful discussions about ML, effects and operational reasoning techniques.

References

[1] P. N. Benton. A unified approach to strictness analysis and optimising transformations. Technical Report 388, Computer Laboratory, University of Cambridge, February 1996.

[2] P. N. Benton, A. J. Kennedy, and G. Russell. Compiling Standard ML to Java bytecodes. In *3rd ACM SIGPLAN International Conference on Functional Programming*, September 1998.

[3] D. K. Gifford and J. M. Lucassen. Integrating functional and imperative programming. In *ACM Conference on Functional Programming and Computer Architecture*, Cambridge, Massachusetts, August 1986.

[4] A. Gordon. *Functional Programming and Input/Output*. PhD thesis, University of Cambridge, 1992. Published as Distinguished Dissertation in Computer Science, CUP, 1994.

[5] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, February 1996.

[6] S. L. Peyton Jones, J. Launchbury, M. B. Shields, and A. P. Tolmach. Bridging the gulf: a common intermediate language for ML and Haskell. In *ACM Symposium on Principles of Programming Languages*, 1998.

[7] S. Peyton Jones and K. Wansbrough. Once upon a polymorphic type. In *ACM Symposium on Principles of Programming Languages*, pages 15–28, 1999.

[8] I. Mason and C. L. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1:287–327, 1991.

[9] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[10] A. M. Pitts. Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute, pages 241–298. Cambridge University Press, 1997.

[11] A. M. Pitts. Reasoning about local variables with operationally-based logical relations. In P. W. O'Hearn and R. D. Tennent, editors, *Algol-Like Languages*, volume 2, chapter 17, pages 173–193. Birkhauser, 1997. Reprinted from *Proceedings Eleventh Annual IEEE Symposium on Logic in Computer Science*, Brunswick, NJ, July 1996, pp 152–163.

[12] A. M. Pitts. Parametric polymorphism and operational equivalence. Technical Report 453, Cambridge University Computer Laboratory, 1998. A preliminary

version appeared in *Proceedings, Second Workshop on Higher Order Operational Techniques in Semantics (HOOTS II), Stanford CA, December 1997*, Electronic Notes in Theoretical Computer Science 10, 1998.

[13] A. M. Pitts and I. D. B. Stark. Operational reasoning for functions with local state. In A. D. Gordon and A. M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, Publications of the Newton Institute, pages 227–273. Cambridge University Press, 1998.

[14] J.-P. Talpin and P. Jouvelot. The type and effect discipline. *Information and Computation*, 111(2), June 1994. Revised from LICS 1992.

[15] A. Tolmach. Optimizing ML using a hierarchy of monadic types. In *Workshop on Types in Compilation*, volume 1473 of *Lecture Notes in Computer Science*, pages 97–113, 1998.

[16] P. Wadler. The marriage of effects and monads. In *International Conference on Functional Programming*, pages 63–74, Baltimore, September 1998. ACM.