Multi-Stage Imperative Languages: A Conservative Extension Result

Cristiano Calcagno[1] and Eugenio Moggi[1*]

DISI, Univ. di Genova, Genova, Italy
{calcagno,moggi}@disi.unige.it

Abstract. This paper extends the recent work [CMT00] on the operational semantics and type system for a core language, called $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$, which exploits the notion of *closed type* (see also [MTBS99]) to safely combine imperative and multi-stage programming. The main novelties are the identification of a larger set of closed types and the addition of a binder for useless variables. The resulting language is a conservative extension of $\mathsf{MiniML}_{\mathsf{ref}}$, a simple imperative subset of SML.

1 Introduction

This paper extends recent work [CMT00] on the operational semantics and type system for a core language, called $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$, which exploits the notion of *closed type* (see also [MTBS99]) to safely combining imperative and multi-stage programming. One would expect that the addition of staging constructs to an imperative language should not prevent writing programs like those in normal imperative languages. In fact, a practical multi-stage programming language like MetaML [Met00] is designed to be *a conservative extension* of a standard programming language, like SML, for good pragmatic reasons: to gain acceptance from an existing user community, and to confine the challenges for new users to the staging constructs only.

Unfortunately, $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$ fails to be a conservative extension of a simple imperative language like $\mathsf{MiniML}_{\mathsf{ref}}$ (i.e. MiniML with ML-style references), because certain well-typed programs in $\mathsf{MiniML}_{\mathsf{ref}}$ fail to be well-typed in $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$. Technically, the problem is that the *closed types* of $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$ are not closed under function types (and that locations may store only values of closed types). The best one can do is to define a translation $\_*$ from $\mathsf{MiniML}_{\mathsf{ref}}$ to $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$ respecting typing and operational semantics. The translation uses the closed type constructor $[\_]$ and closedness annotations, in particular the translation of a functional type is $(t_1 \to t_2)* \stackrel{\Delta}{=} [t_1* \to t_2*]$, which records that a functional type in the source language is a *closed functional type* in the target language.

From a language design perspective the main contribution of this paper is a core language, called $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$, which extends *conservatively* the simple imperative language $\mathsf{MiniML}_{\mathsf{ref}}$ with the staging constructs of MetaML (and a few

---

* Research partially supported by MURST and ESPRIT WG APPSEM.

other features related to closed types). A safe combination of imperative and multi-stage programming in $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ is enforced through the use of closed types, as done in [CMT00] for $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$.

Technically, the main novelty over [CMT00] is the identification of a larger set of closed types, which includes functional types of the form $t \to c$ where c is a closed type. The closed types of $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$ enjoy the following property: values of closed types are closed, i.e. have no free variables. The closed types of $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ enjoy a weaker property (which is the best one can hope for functional types): the free variables in values of closed types are *useless*, i.e. during evaluation they will never be evaluated (at level 0).

Examples. The restriction of storable values to closed types is motivated by the following MetaML session:

```
-| val l = ref <1>;
val l = ... : ref <int>
-| val f = <fn x => ~(l:=<x>; <2>)>;
val f = <fn x => 2> : <int -> int>
-| val c = !l;
val c = <x> : <int>
```

In evaluating the second declaration, the variable `x` goes outside the scope of the binding lambda, and the result of the third line is wrong, since x is not bound in the environment, even though the session is well-typed according to naive extensions of previously proposed type systems for MetaML. This form of **scope extrusion** is specific to multi-level and multi-stage languages, and it does *not* arise in traditional programming languages, where evaluation is generally restricted to closed terms. The problem lies in the *run-time interaction between free variables and references.* In the type system we propose the above session is not well-typed: `l:=<x>` cannot be typed, because `<x>` is not of a closed type.

$\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ allows among the closed types some functional types, while in $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$ functional types are never closed. The following interactive session, is typable in $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ but not in $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$.

```
-| val l = ref(fn x => x+1);
val l = ... : (int -> int) ref
-| val f = <fn x => ~(l := (fn y => ((fn z => y+1) <x>)); <x+1>)>;
val f = <(fn x => x+1)> : <int -> int>
```

The first line creates a reference to functions from integers to integers; and the second assigns the function `fn y => ((fn z => y+1) <x>)` to it. As a result, the variable `x` escapes from its binder and leaks into the store. However, this cannot be observed because the variable is "useless": if we supply an argument to the stored function, the inner application will be evaluated, discarding the term `<x>`. The operational semantics presented here solves the problem with a binder for useless variables, introduced before storing a term.

Relation to $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$. There is a significant overlap between $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ and $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$. We refer to [CMT00] for a broader discussion of related work [DP96,Dav96,TS97,TBS98,MTBS99,BMTS99,Tah99,TS00]. For those familiar with $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$ (recalled in Appendix A) we summarize the differences:

- $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ has no closedness annotation $[e]$, and the closed type constructor $[\_]$ cannot be applied to a closed type c. These are *cosmetic* changes, motivated by the following remarks in [CMT00]: closedness annotations play no role in the operational semantics, and a closed type c is *semantically* isomorphic to $[c]$ via the mapping $x \mapsto [x]$. When closedness annotations are removed, the isomorphism becomes an identity, thus the syntax for $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ types *forbids* $[c]$, since it is *equal* to c.
- $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ has a let-binder $(\mathsf{let}_c x = e_1 \,\mathsf{in}\, e_2)$ corresponding to $(\mathsf{let}\,[x{:}\,c] = [e_1]\,\mathsf{in}\, e_2)$ of $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$, for variables of closed type.
- $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ has a larger set of closed types, in particular a functional type $t \to c$ is closed whenever c is closed. This property is essential to prove that every well-formed $\mathsf{MiniML}_{\mathsf{ref}}$ program is also well-formed in $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$.
- $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ has a new binder $\bullet e$, called Bullet, which binds all the free variables in e. When all the free variables in e are *useless*, $\bullet e$ and e are *semantically* equivalent. Bullet is used in the operational semantics to prevent scope extrusion (for this purpose it replaces the constant fault of [CMT00]), and to annotate terms whose free variables are useless.
  In an implementation, Bullet should help improve efficiency, since one knows that $\mathrm{FV}(\bullet e) = \emptyset$ without examining the whole of e. For instance, the function $\bullet\lambda x.e$ does not depend on the environment, only on the argument. Our operational semantics is too abstract to support claims about efficiency, but we expect that a reformulation in terms of weak explicit substitution ([LM99,B97]) could make such claims precise.
  In general, checking whether a variable is useless requires a static analysis (preferably of the whole program, see [WS99]). The $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ type system has a simple rule to infer $\bullet e{:}\, c^n$, namely $e{:}\, c^n$ when all the free variables in e have level $> n$. This rule makes sense only in the context of multi-level languages, but it infers $\bullet(!\,l\ \langle x\rangle){:}\, c^n$, where l is a location of closed type $\langle t\rangle \to c$, which is beyond conventional analyses.

Structure of the paper. Section 2 introduces $\mathsf{MiniML}_{\mathsf{ref}}$, which is MiniML of [CDDK86] with ML-style references. Section 3 introduces $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$, which extends $\mathsf{MiniML}_{\mathsf{ref}}$ with

- The three staging constructs of MetaML [TS97,TBS98,Met00]: Brackets $\langle e\rangle$, Escape $\tilde{}e$ and Run $\mathsf{run}\ e$.
- A let-binder $(\mathsf{let}_c x = e_1 \,\mathsf{in}\, e_2)$ for variables of closed type.
- A binder $\bullet e$, called Bullet, of all the free variables in a term e of closed type.

We also prove type safety along the lines of [CMT00]. Section 4 shows that $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ is a conservative extension of $\mathsf{MiniML}_{\mathsf{ref}}$. Section 5 discusses improvements to the type system through the addition of sub-typing, alternatives to Bullet, and variation to the syntax and operational semantics of $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$.

2 $\mathsf{MiniML_{ref}}$

This section describes the syntax, type system and operational semantics of $\mathsf{MiniML_{ref}}$, an extension of MiniML ([CDDK86]) with ML-style references. Types t are defined as

$$t \in \mathsf{T}\colon := \mathsf{nat} \mid \mathsf{ref}\, t \mid t_1 \to t_2$$

The sets of $\mathsf{MiniML_{ref}}$ terms and values are parametric in an infinite set of variables $x \in \mathsf{X}$ and an infinite set of locations $l \in \mathsf{L}$

$$\begin{aligned} e \in \mathsf{E}\colon := {} & x \mid \lambda x.e \mid e_1 e_2 \mid \mathsf{fix}\, x.e \mid \mathsf{z} \mid \mathsf{s}\, e \mid (\mathsf{case}\, e\, \mathsf{of}\, \mathsf{z} \to e_1 \mid \mathsf{s}\, x \to e_2) \mid \\ & \mathsf{ref}\, e \mid !\, e \mid e_1 := e_2 \mid l \\ v \in \mathsf{V}\colon := {} & \lambda x.e \mid \mathsf{z} \mid \mathsf{s}\, v \mid l \end{aligned}$$

The first line lists the MiniML terms: variables, abstraction, application, fix-point for recursive definitions, zero, successor, and case-analysis on natural numbers. The second line lists the three SML operations on references, and constants l for locations. These constants are not allowed in user-defined programs, but they are instrumental to the operational semantics of $\mathsf{MiniML_{ref}}$.

Note 1. We will use the following notation and terminology

- Term equivalence, written $\equiv$, is α-conversion. $\mathrm{FV}(e)$ is the set of variables free in e. E_0 indicates the set of terms without free variables. Substitution of e_1 for x in e_2 (modulo $\equiv$) is written $e_2[x := e_1]$.
- m, n range over the set N of natural numbers. Furthermore, $m \in \mathsf{N}$ is identified with the set $\{i \in \mathsf{N} | i < m\}$ of its predecessors.
- $f\colon A \stackrel{fin}{\to} B$ means that f is a partial function from A to B with a finite domain, written $dom(f)$.
- $\Sigma\colon \mathsf{L} \stackrel{fin}{\to} \mathsf{T}$ is a *signature* (for locations only), written $\{l_i\colon \mathsf{ref}\, t_i | i \in m\}$.
- $\Gamma\colon \mathsf{X} \stackrel{fin}{\to} \mathsf{T}$ is a type assignment, written $\{x_i\colon t_i | i \in m\}$.
- $\mu \in \mathsf{S} \stackrel{\Delta}{=} \mathsf{L} \stackrel{fin}{\to} \mathsf{V}_0$ is a store, where V_0 is the set of closed values.
- $\Sigma, l\colon \mathsf{ref}\, t$, $\Gamma, x\colon t$ and $\mu\{l = v\}$ denote extension/update of a signature, assignment and store respectively.

Type System. The type system of $\mathsf{MiniML_{ref}}$ is given in Figure 1, and it enjoys the following basic properties:

Lemma 1 (Weakening).

1. $\Sigma; \Gamma \vdash e\colon t_2$ *and* x fresh *imply* $\Sigma; \Gamma, x\colon t_1 \vdash e\colon t_2$
2. $\Sigma; \Gamma \vdash e\colon t_2$ *and* l fresh *imply* $\Sigma, l\colon \mathsf{ref}\, t_1; \Gamma \vdash e\colon t_2$

Lemma 2 (Substitution).
$\Sigma; \Gamma \vdash e_1\colon t_1$ *and* $\Sigma; \Gamma, x\colon t_1 \vdash e_2\colon t_2$ *imply* $\Sigma; \Gamma \vdash e_2[x := e_1]\colon t_2$

We say that a store μ is well-formed for Σ (and write $\Sigma \models \mu$) $\stackrel{\Delta}{\Longleftrightarrow}$

$$dom(\Sigma) = dom(\mu) \text{ and } \Sigma \vdash v\colon t \text{ whenever } \mu(l) = v \text{ and } \Sigma(l) = \mathsf{ref}\, t.$$

$$\frac{}{\Sigma;\Gamma\vdash x\colon t}\,\Gamma(x)=t \qquad \frac{\Sigma;\Gamma,x\colon t_1\vdash e\colon t_2}{\Sigma;\Gamma\vdash \lambda x.e\colon t_1\to t_2} \qquad \frac{\Sigma;\Gamma\vdash e_1\colon t_1\to t_2 \quad \Sigma;\Gamma\vdash e_2\colon t_1}{\Sigma;\Gamma\vdash e_1e_2\colon t_2}$$

$$\frac{\Sigma;\Gamma,x\colon t\vdash e\colon t}{\Sigma;\Gamma\vdash \mathsf{fix}\,x.e\colon t} \qquad \frac{}{\Sigma;\Gamma\vdash \mathsf{z}\colon \mathsf{nat}} \qquad \frac{\Sigma;\Gamma\vdash e\colon \mathsf{nat}}{\Sigma;\Gamma\vdash \mathsf{s}\,e\colon \mathsf{nat}}$$

$$\frac{\Sigma;\Gamma\vdash e\colon \mathsf{nat} \quad \Sigma;\Gamma\vdash e_1\colon t \quad \Sigma;\Gamma,x\colon\mathsf{nat}\vdash e_2\colon t}{\Sigma;\Gamma\vdash (\mathsf{case}\,e\,\mathsf{of}\,\mathsf{z}\to e_1 \mid \mathsf{s}\,x\to e_2)\colon t} \qquad \frac{\Sigma;\Gamma\vdash e\colon t}{\Sigma;\Gamma\vdash \mathsf{ref}\,e\colon \mathsf{ref}\,t}$$

$$\frac{\Sigma;\Gamma\vdash e\colon \mathsf{ref}\,t}{\Sigma;\Gamma\vdash\, !\,e\colon t} \qquad \frac{\Sigma;\Gamma\vdash e_1\colon \mathsf{ref}\,t \quad \Sigma;\Gamma\vdash e_2\colon t}{\Sigma;\Gamma\vdash e_1 := e_2\colon \mathsf{ref}\,t} \qquad \frac{}{\Sigma;\Gamma\vdash l\colon \mathsf{ref}\,t}\,\Sigma(l)=\mathsf{ref}\,t$$

Fig. 1. Type System for $\mathsf{MiniML_{ref}}$

Operational Semantics. The operational semantics of $\mathsf{MiniML_{ref}}$ is given in Figure 2. The semantics is non-deterministic because of the rule for evaluating $\mathsf{ref}\,e$. Evaluation of a term $e \in \mathsf{E}_0$ with an initial store μ_0 can lead to

- a *result* v and a new store μ_1, when we can derive $\mu_0, e \hookrightarrow \mu_1, v$, or
- a *run-time error*, when we can derive $\mu_0, e \hookrightarrow \mathsf{err}$.

Evaluation of a term may also lead to divergence, although a big-step operational semantics can express this third possibility only indirectly. One would have to adopt a reduction semantics (as advocated by [WF94]) to achieve a more accurate classification of the possible computations. In our setting, Type Safety means that evaluation of a well-typed program cannot lead to a run-time error, namely

Theorem 1 (Safety). *$\mu_0, e \hookrightarrow d$ and $\Sigma_0 \models \mu_0$ and $\Sigma_0 \vdash e\colon t$ imply that there exist μ_1 and v and Σ_1 such that $d \equiv (\mu_1, v)$ and $\Sigma_0, \Sigma_1 \models \mu_1$ and $\Sigma_0, \Sigma_1 \vdash v\colon t$.*

3 $\mathsf{MiniML^{meta}_{ref}}$

This section describes the syntax, type system and operational semantics of $\mathsf{MiniML^{meta}_{ref}}$, and establishes Type Safety. Types t, closed types c and open types o are defined as

$$\begin{aligned} t \in \mathsf{T} &::= c \mid o \\ c \in \mathsf{C} &::= \mathsf{nat} \mid t\to c \mid [o] \mid \mathsf{ref}\,c \\ o \in \mathsf{O} &::= t\to o \mid \langle t\rangle \end{aligned}$$

Intuitively, a term can be assigned a closed type c only when its free variables are *useless*. The set of $\mathsf{MiniML^{meta}_{ref}}$ terms is parametric in an infinite set of variables $x \in \mathsf{X}$ and an infinite set of locations $l \in \mathsf{L}$

$$\begin{aligned} e \in \mathsf{E} ::= {} & x \mid \lambda x.e \mid e_1e_2 \mid \mathsf{fix}\,x.e \mid \mathsf{z} \mid \mathsf{s}\,e \mid (\mathsf{case}\,e\,\mathsf{of}\,\mathsf{z}\to e_1 \mid \mathsf{s}\,x\to e_2) \mid \\ & \langle e\rangle \mid \tilde{}\,e \mid \mathsf{run}\;e \mid (\mathsf{let}_c x = e_1\,\mathsf{in}\,e_2) \mid \bullet\, e \mid \\ & \mathsf{ref}\,e \mid !\,e \mid e_1 := e_2 \mid l \end{aligned}$$

$$\mu_0, \lambda x.e \hookrightarrow \mu_0, \lambda x.e \qquad \frac{\mu_0, e_1 \hookrightarrow \mu_1, \lambda x.e \quad \mu_1, e_2 \hookrightarrow \mu_2, v_2 \quad \mu_2, e[x{:}= v_2] \hookrightarrow \mu_3, v}{\mu_0, e_1 e_2 \hookrightarrow \mu_3, v}$$

$$\frac{\mu_0, e_1 \hookrightarrow \mu_1, v \not\equiv \lambda x.e}{\mu_0, e_1 e_2 \hookrightarrow \mathsf{err}} \quad \frac{\mu_0, e[x{:}= \mathsf{fix}\, x.e] \hookrightarrow \mu_1, v}{\mu_0, \mathsf{fix}\, x.e \hookrightarrow \mu_1, v} \quad \mu_0, \mathsf{z} \hookrightarrow \mu_0, \mathsf{z}$$

$$\frac{\mu_0, e \hookrightarrow \mu_1, v}{\mu_0, \mathsf{s}\, e \hookrightarrow \mu_1, \mathsf{s}\, v} \quad \frac{\mu_0, e \hookrightarrow \mu_1, \mathsf{z} \quad \mu_1, e_1 \hookrightarrow \mu_2, v}{\mu_0, (\mathsf{case}\, e\, \mathsf{of}\, \mathsf{z} \to e_1 \mid \mathsf{s}\, x \to e_2) \hookrightarrow \mu_2, v}$$

$$\frac{\mu_0, e \hookrightarrow \mu_1, v \not\equiv \mathsf{z} \mid \mathsf{s}\, e'}{\mu_0, (\mathsf{case}\, e\, \mathsf{of}\, \mathsf{z} \to e_1 \mid \mathsf{s}\, x \to e_2) \hookrightarrow \mathsf{err}} \quad \frac{\mu_0, e \hookrightarrow \mu_1, \mathsf{s}\, v \quad \mu_1, e_2[x{:}= v] \hookrightarrow \mu_2, v_2}{\mu_0, (\mathsf{case}\, e\, \mathsf{of}\, \mathsf{z} \to e_1 \mid \mathsf{s}\, x \to e_2) \hookrightarrow \mu_2, v_2}$$

$$\frac{\mu_0, e \hookrightarrow \mu_1, v}{\mu_0, \mathsf{ref}\, e \hookrightarrow \mu_1\{l = v\}, l}\; l \notin dom(\mu_1) \quad \frac{\mu_0, e \hookrightarrow \mu_1, l}{\mu_0, !\, e \hookrightarrow \mu_1, v}\; \mu_1(l) \equiv v$$

$$\frac{\mu_0, e \hookrightarrow \mu_1, v \not\equiv l \in dom(\mu_1)}{\mu_0, !\, e \hookrightarrow \mathsf{err}} \quad \frac{\mu_0, e_1 \hookrightarrow \mu_1, l \quad \mu_1, e_2 \hookrightarrow \mu_2, v}{\mu_0, e_1{:}= e_2 \hookrightarrow \mu_2\{l = v\}, l}$$

$$\frac{\mu_0, e_1 \hookrightarrow \mu_1, v \not\equiv l \in dom(\mu_1)}{\mu_0, e_1{:}= e_2 \hookrightarrow \mathsf{err}} \quad \mu_0, l \hookrightarrow \mu_0, l$$

The rules for error propagation follow the ML-convention, i.e. for every normal evaluation rule $\dfrac{\{\mu_i, e_i \hookrightarrow \mu_{i+1}, v_i \mid i \in n\}}{\mu_0, e \hookrightarrow \mu_n, v}$ and every $m \in n$ one should add an error propagation rule $\dfrac{\{\mu_i, e_i \hookrightarrow \mu_{i+1}, v_i \mid i \in m\} \quad \mu_m, e_m \hookrightarrow \mathsf{err}}{\mu_0, e \hookrightarrow \mathsf{err}}$.

Fig. 2. Operational Semantics for $\mathsf{MiniML}_{\mathsf{ref}}$

The second line lists the three multi-stage constructs of MetaML [TS97]: *Brackets* $\langle e \rangle$ and *Escape* $\tilde{}\, e$ are for building and splicing code, and *Run* is for executing code. The second line lists also a let-binder ($\mathsf{let}_c x = e_1\, \mathsf{in}\, e_2$) for variables of closed type, and a binder *Bullet* $\bullet e$, which binds all the free variables of e, hence $FV(\bullet e) = \emptyset$, and $(\bullet e)[x{:}= e_1] \equiv \bullet e$.

Note 2. We will use the following notation and terminology (see also Note 1)

- w ranges over terms not of the form $\bullet e$, while $\circ w$ can be either w of $\bullet w$.
- $\Sigma{:}\, \mathsf{L} \stackrel{fin}{\to} \mathsf{T}$ is a *signature* (for locations only), written $\{l_i{:}\, \mathsf{ref}\, c_i | i \in m\}$.
- $\Delta{:}\, \mathsf{X} \stackrel{fin}{\to} (\mathsf{C} \times \mathsf{N})$ and $\Gamma{:}\, \mathsf{X} \stackrel{fin}{\to} (\mathsf{T} \times \mathsf{N})$ are type-and-level assignments, written $\{x_i{:}\, c_i^{n_i} | i \in m\}$ and $\{x_i{:}\, t_i^{n_i} | i \in m\}$ respectively.
  We use the following operations on type-and-level assignments:
  $\{x_i{:}\, t_i^{n_i} | i \in m\}^{+n} \stackrel{\Delta}{=} \{x_i{:}\, t_i^{n_i+n} | i \in m\}$ adds n to the level of the x_i;
  $\{x_i{:}\, t_i^{n_i} | i \in m\}^{\leq n} \stackrel{\Delta}{=} \{x_i{:}\, t_i^{n_i} | n_i \leq n \wedge i \in m\}$ removes the x_i with level $> n$.
- $\Gamma, x{:}\, t^n$ and $\Delta, x{:}\, c^n$ denote the extension of type-and-level assignments.

Remark 1. The new binder Bullet $\bullet e$ serves many purposes, which the constant fault of [CMT00] can fulfill only in part (e.g. fault is not typable). Intuitively, $\bullet e$ is like a closure (e, ρ), where ρ is the environment (explicit substitution) mapping all variables to fault, and in addition it records that e should have a closed type.

$$\frac{}{\Sigma;\Delta;\Gamma\vdash x\colon t^n}\ (\Delta,\Gamma)(x)=t^n \qquad \frac{\Sigma;\Delta;\Gamma,x\colon t_1^n\vdash e\colon t_2^n}{\Sigma;\Delta;\Gamma\vdash \lambda x.e\colon t_1\to t_2^n}$$

$$\frac{\Sigma;\Delta;\Gamma\vdash e_1\colon t_1\to t_2^n \quad \Sigma;\Delta;\Gamma\vdash e_2\colon t_1^n}{\Sigma;\Delta;\Gamma\vdash e_1 e_2\colon t_2^n} \qquad \frac{\Sigma;\Delta;\Gamma,x\colon t^n\vdash e\colon t^n}{\Sigma;\Delta;\Gamma\vdash \mathsf{fix}\,x.e\colon t^n} \qquad \frac{}{\Sigma;\Delta;\Gamma\vdash \mathsf{z}\colon \mathsf{nat}^n}$$

$$\frac{\Sigma;\Delta;\Gamma\vdash e\colon \mathsf{nat}^n}{\Sigma;\Delta;\Gamma\vdash \mathsf{s}\,e\colon \mathsf{nat}^n} \qquad \frac{\Sigma;\Delta;\Gamma\vdash e\colon \mathsf{nat}^n \quad \Sigma;\Delta;\Gamma\vdash e_1\colon t^n \quad \Sigma;\Delta;\Gamma,x\colon \mathsf{nat}^n\vdash e_2\colon t^n}{\Sigma;\Delta;\Gamma\vdash (\mathsf{case}\,e\,\mathsf{of}\,\mathsf{z}\to e_1 \mid \mathsf{s}\,x\to e_2)\colon t^n}$$

$$\frac{\Sigma;\Delta;\Gamma\vdash e\colon c^n}{\Sigma;\Delta;\Gamma\vdash \mathsf{ref}\,e\colon \mathsf{ref}\,c^n} \qquad \frac{\Sigma;\Delta;\Gamma\vdash e\colon \mathsf{ref}\,c^n}{\Sigma;\Delta;\Gamma\vdash\, !\,e\colon c^n} \qquad \frac{\Sigma;\Delta;\Gamma\vdash e_1\colon \mathsf{ref}\,c^n \quad \Sigma;\Delta;\Gamma\vdash e_2\colon c^n}{\Sigma;\Delta;\Gamma\vdash e_1 := e_2\colon \mathsf{ref}\,c^n}$$

$$\frac{}{\Sigma;\Delta;\Gamma\vdash l\colon \mathsf{ref}\,c^n}\ \Sigma(l)=\mathsf{ref}\,c \qquad \frac{\Sigma;\Delta;\Gamma\vdash e\colon t^{n+1}}{\Sigma;\Delta;\Gamma\vdash \langle e\rangle\colon \langle t\rangle^n} \qquad \frac{\Sigma;\Delta;\Gamma\vdash e\colon \langle t\rangle^n}{\Sigma;\Delta;\Gamma\vdash \,\tilde{}\,e\colon t^{n+1}}$$

$$\frac{\Sigma;\Delta;\Gamma\vdash e\colon [\langle t\rangle]^n}{\Sigma;\Delta;\Gamma\vdash \mathsf{run}\,e\colon t^n} \qquad \frac{\Sigma;\Delta^{\leq n};\emptyset\vdash e\colon o^n}{\Sigma;\Delta;\Gamma\vdash e\colon [o]^n} \qquad \frac{\Sigma;\Delta;\Gamma\vdash e\colon [o]^n}{\Sigma;\Delta;\Gamma\vdash e\colon o^n}$$

$$\frac{\Sigma;\Delta;\Gamma\vdash e_1\colon c^n \quad \Sigma;\Delta,x\colon c^n;\Gamma\vdash e_2\colon t^n}{\Sigma;\Delta;\Gamma\vdash (\mathsf{let}_c x = e_1\ \mathsf{in}\ e_2)\colon t^n} \qquad \frac{\Sigma;\Delta_1^{+(n+1)};\Gamma_1^{+(n+1)}\vdash e\colon c^n}{\Sigma;\Delta;\Gamma\vdash \bullet e\colon c^n}$$

Fig. 3. Type System for $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$

The typing rule for Bullet, in combination with Type Safety (Theorem 2), formalizes the property that in a term of closed type (at level n) all the free variables (at level $> n$) are *useless.* In fact, during evaluation a variable bound by Bullet (unlike variables captured by other binders) cannot get instantiated, thus its occurrences must disappear before reaching level 0 (otherwise a run-time error will occur).

The operational semantics of Figure 2 uses Bullet to prevent scope extrusion when a location l is initialized or assigned. In fact, what gets stored in l is the closed value $\bullet w$, instead of the value w. Therefore, if a free variable in w was is the scope of an enclosing binder, e.g. x in $\langle \lambda x.\tilde{}(l := w; \langle x\rangle)\rangle$, it is caught by Bullet, instead of becoming free.

Unlike locations (which exist only at execution time) and fault (which is not typable), Bullet could be used in user-defined programs to record that a term has a closed type. The operational semantics uses such information when evaluating an application (if $\lambda x.e$ has a closed type, then e must have a closed type) and a let-binder (the let must bind x to a term of closed type) for capturing free variables. For instance, during evaluation of $\bullet(\lambda x.e)\ v$ the free variables of v get captured in $\bullet(e[x := v])$.

3.1 Type System

Figure 3 gives the type system of $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$. A typing judgement has the form $\Sigma;\Delta;\Gamma\vdash e\colon t^n$, read *"$e$ has type t and level n under the assignment $\Sigma;\Delta;\Gamma$"*. Σ gives the type of locations which can be used in e, Δ and Γ (must have disjoint domains and) give the type and level of variables which may occur free in e.

Remark 2. All typing rules, except the last four, are borrowed from [CMT00]. The introduction and elimination rules for $[o]$ say that $[o]$ is a sub-type of o. The rule for $(\mathsf{let}_c x = e_1 \,\mathsf{in}\, e_2)$ incorporates the typing rule (close*) of [CMT00]. The rule for $\bullet e$ says that Bullet binds all the free variables in e. One can think of $\bullet e$ as the closure (e, ρ), where ρ is the environment (explicit substitution) mapping all variables to fault.

The type system enjoys the following basic properties (see also [CMT00]):

Lemma 3 (Weakening).

1. $\Sigma; \Delta; \Gamma \vdash e\colon t_2^n$ *and* x fresh *imply* $\Sigma; \Delta; \Gamma, x\colon t_1^m \vdash e\colon t_2^n$
2. $\Sigma; \Delta; \Gamma \vdash e\colon t_2^n$ *and* x fresh *imply* $\Sigma; \Delta, x\colon c_1^m; \Gamma \vdash e\colon t_2^n$
3. $\Sigma; \Delta; \Gamma \vdash e\colon t_2^n$ *and* l fresh *imply* $\Sigma, l\colon \mathsf{ref}\, c_1; \Delta; \Gamma \vdash e\colon t_2^n$

Lemma 4 (Substitution).

1. $\Sigma; \Delta; \Gamma \vdash e_1\colon t_1^m$ *and* $\Sigma; \Delta; \Gamma, x\colon t_1^m \vdash e_2\colon t_2^n$ *imply* $\Sigma; \Delta; \Gamma \vdash e_2[x\colon= e_1]\colon t_2^n$
2. $\Sigma; \Delta^{\leq m}; \emptyset \vdash e_1\colon c_1^m$ *and* $\Sigma; \Delta, x\colon c_1^m; \Gamma \vdash e_2\colon t_2^n$ *imply* $\Sigma; \Delta; \Gamma \vdash e_2[x\colon= e_1]\colon t_2^n$

3.2 Operational Semantics

The operational semantics of $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ is given in Figure 4. The rules derive evaluation judgements of the form $\mu, e \overset{n}{\hookrightarrow} d$, where $\mu \in \mathsf{S}$ is a *value store* (see below). In the rules v ranges over terms, but *a posteriori* one can show that v ranges over *values at level* n (see below). We will show that evaluation of a well-typed program cannot lead to a run-time error (Theorem 2).

Definition 1. *The set* $\mathsf{V}^n \subset \mathsf{E}$ *of* **values at level** n *is defined by the BNF*

$$
\begin{array}{rl}
v^n \in \mathsf{V}^n ::= & w^n \mid \bullet\, w^n \\
w^0 \in \mathsf{W}^0 ::= & \lambda x.e \mid \mathsf{z} \mid \mathsf{s}\, v^0 \mid \langle v^1 \rangle \mid l \\
w^{n+1} \in \mathsf{W}^{n+1} ::= & x \mid \lambda x.v^{n+1} \mid v_1^{n+1} v_2^{n+1} \mid \mathsf{fix}\, x.v^{n+1} \mid \mathsf{z} \mid \mathsf{s}\, v^{n+1} \mid \\
& (\mathsf{case}\, v^{n+1}\, \mathsf{of}\, \mathsf{z} \to v_1^{n+1} \mid \mathsf{s}\, x \to v_2^{n+1}) \mid \langle v^{n+2} \rangle \mid \mathsf{run}\, v^{n+1} \mid \\
& (\mathsf{let}_c x = v_1^{n+1}\, \mathsf{in}\, v_2^{n+1}) \mid \\
& \mathsf{ref}\, v^{n+1} \mid\, !\, v^{n+1} \mid v_1^{n+1} \colon= v_2^{n+1} \mid l \\
w^{n+2} \in \mathsf{W}^{n+2} += & \text{\textasciitilde} v^{n+1}
\end{array}
$$

$\mu \in \mathsf{S} \overset{\Delta}{=} \mathsf{L} \overset{fin}{\to} \mathsf{V}_0^0$ *is a* **value store**, *where* V_0^0 *is the set of closed values at level* 0. *We write* $\Sigma \models \mu \overset{\Delta}{\Longleftrightarrow} dom(\Sigma) = dom(\mu)$ *and* $\Sigma; \emptyset \vdash v\colon c^0$ *whenever* $\mu(l) = v$ *and* $\Sigma(l) = \mathsf{ref}\, c$.

The following result establishes basic facts about the operational semantics, similar to those established for $\mathsf{MiniML}^{\mathsf{BN}}_{\mathsf{ref}}$ (see [CMT00]).

Lemma 5 (Values). $\mu_0, e \overset{n}{\hookrightarrow} \mu_1, v$ *and* μ_0 *is value store imply* μ_1 *is a value store,* $dom(\mu_0) \subseteq dom(\mu_1)$, $v \in \mathsf{V}^n$ *and* $\mathrm{FV}(v) \subseteq \mathrm{FV}(e)$.

In the rules below $\circ w$ is a meta-expression ranging over terms of the form w and $\bullet w$.

Normal Evaluation

$$\mu_0, \lambda x.e \overset{0}{\hookrightarrow} \mu_0, \lambda x.e \qquad \frac{\mu_0, e_1 \overset{0}{\hookrightarrow} \mu_1, \lambda x.e \quad \mu_1, e_2 \overset{0}{\hookrightarrow} \mu_2, v_2 \quad \mu_2, e[x{:}= v_2] \overset{0}{\hookrightarrow} \mu_3, v}{\mu_0, e_1 e_2 \overset{0}{\hookrightarrow} \mu_3, v}$$

$$\mu_0, x \overset{0}{\hookrightarrow} \mathsf{err} \qquad \frac{\mu_0, e_1 \overset{0}{\hookrightarrow} \mu_1, \bullet\lambda x.e \quad \mu_1, e_2 \overset{0}{\hookrightarrow} \mu_2, v_2 \quad \mu_2, \bullet(e[x{:}= v_2]) \overset{0}{\hookrightarrow} \mu_3, v}{\mu_0, e_1 e_2 \overset{0}{\hookrightarrow} \mu_3, v}$$

$$\frac{\mu_0, e_1 \overset{0}{\hookrightarrow} \mu_1, v \not\equiv \circ\lambda x.e}{\mu_0, e_1 e_2 \overset{0}{\hookrightarrow} \mathsf{err}} \qquad \frac{\mu_0, e[x{:}= \mathsf{fix}\, x.e] \overset{0}{\hookrightarrow} \mu_1, v}{\mu_0, \mathsf{fix}\, x.e \overset{0}{\hookrightarrow} \mu_1, v} \qquad \mu_0, \mathsf{z} \overset{0}{\hookrightarrow} \mu_0, \mathsf{z}$$

$$\frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, v}{\mu_0, \mathsf{s}\, e \overset{0}{\hookrightarrow} \mu_1, \mathsf{s}\, v} \qquad \frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, \circ\mathsf{z} \quad \mu_1, e_1 \overset{0}{\hookrightarrow} \mu_2, v_1}{\mu_0, (\mathsf{case}\, e\, \mathsf{of}\, \mathsf{z} \to e_1 \mid \mathsf{s}\, x \to e_2) \overset{0}{\hookrightarrow} \mu_2, v_1}$$

$$\frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, v \not\equiv \circ\mathsf{z} \mid \circ\, \mathsf{s}\, v}{\mu_0, (\mathsf{case}\, e\, \mathsf{of}\, \mathsf{z} \to e_1 \mid \mathsf{s}\, x \to e_2) \overset{0}{\hookrightarrow} \mathsf{err}} \qquad \frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, \circ\mathsf{s}\, v \quad \mu_1, e_2[x{:}= \circ v] \overset{0}{\hookrightarrow} \mu_2, v_2}{\mu_0, (\mathsf{case}\, e\, \mathsf{of}\, \mathsf{z} \to e_1 \mid \mathsf{s}\, x \to e_2) \overset{0}{\hookrightarrow} \mu_2, v_2}$$

$$\frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, \circ w}{\mu_0, \mathsf{ref}\; e \overset{0}{\hookrightarrow} \mu_1\{l = \bullet w\}, l}\; l \notin dom(\mu_1) \qquad \frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, \circ l}{\mu_0, !\, e \overset{0}{\hookrightarrow} \mu_1, v}\; \mu_1(l) \equiv v$$

$$\frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, \circ w}{\mu_0, !\, e \overset{0}{\hookrightarrow} \mathsf{err}}\; w \notin dom(\mu_1) \qquad \frac{\mu_0, e_1 \overset{0}{\hookrightarrow} \mu_1, \circ l \quad \mu_1, e_2 \overset{0}{\hookrightarrow} \mu_2, \circ w}{\mu_0, e_1 {:}= e_2 \overset{0}{\hookrightarrow} \mu_2\{l = \bullet w\}, l}\; l \in dom(\mu_1)$$

$$\frac{\mu_0, e_1 \overset{0}{\hookrightarrow} \mu_1, \circ w}{\mu_0, e_1 {:}= e_2 \overset{0}{\hookrightarrow} \mathsf{err}}\; w \notin dom(\mu_1) \qquad \mu_0, l \overset{0}{\hookrightarrow} \mu_0, l \qquad \frac{\mu_0, e \overset{1}{\hookrightarrow} \mu_1, v}{\mu_0, \langle e \rangle \overset{0}{\hookrightarrow} \mu_1, \langle v \rangle} \qquad \mu_0, {\sim}e \overset{0}{\hookrightarrow} \mathsf{err}$$

$$\frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, \circ\langle v \rangle \quad \mu_1, \bullet v \overset{0}{\hookrightarrow} \mu_2, v_0}{\mu_0, \mathsf{run}\; e \overset{0}{\hookrightarrow} \mu_2, v_0} \qquad \frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, v \not\equiv \circ\langle e' \rangle}{\mu_0, \mathsf{run}\; e \overset{0}{\hookrightarrow} \mathsf{err}}$$

$$\frac{\mu_0, e_1 \overset{0}{\hookrightarrow} \mu_1, \circ w \quad \mu_1, e_2[x{:}= \bullet w] \overset{0}{\hookrightarrow} \mu_2, v_2}{\mu_0, (\mathsf{let}_c x = e_1\, \mathsf{in}\, e_2) \overset{0}{\hookrightarrow} \mu_2, v_2} \qquad \frac{\mu_0, e \overset{n}{\hookrightarrow} \mu_1, \circ w}{\mu_0, \bullet e \overset{n}{\hookrightarrow} \mu_1, \bullet w}$$

Symbolic Evaluation

$$\mu_0, x \overset{n+1}{\hookrightarrow} \mu_0, x \qquad \frac{\mu_0, e \overset{n+2}{\hookrightarrow} \mu_1, v}{\mu_0, \langle e \rangle \overset{n+1}{\hookrightarrow} \mu_1, \langle v \rangle} \qquad \frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, \langle v \rangle}{\mu_0, {\sim}e \overset{1}{\hookrightarrow} \mu_1, v} \qquad \frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, \bullet\langle \circ w \rangle}{\mu_0, {\sim}e \overset{1}{\hookrightarrow} \mu_1, \bullet w}$$

$$\frac{\mu_0, e \overset{0}{\hookrightarrow} \mu_1, v \not\equiv \circ\langle e' \rangle}{\mu_0, {\sim}e \overset{1}{\hookrightarrow} \mathsf{err}} \qquad \frac{\mu_0, e \overset{n+1}{\hookrightarrow} \mu_1, v}{\mu_0, {\sim}e \overset{n+2}{\hookrightarrow} \mu_1, {\sim}v}$$

In all other cases symbolic evaluation is applied to the immediate sub-terms from left to right without changing level.

Error Propagation

The rules for error propagation follow the ML-convention (see Figure 2).

Fig. 4. Operational Semantics for $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$

Proof. By induction on the derivation of the evaluation judgement $\mu_0, e \overset{n}{\hookrightarrow} \mu_1, v$. Notice that in the rules evaluating $\mathsf{ref}\, e$ and $e_1 := e_2$ it is important that we store $\bullet w$, since w may have free variables.

The following lemma is used to prove type safety in the case for evaluating $\mathsf{run}\, e$ at level 0 and $\tilde{}\, e$ at level 1. The result holds also for closed types of the form nat and $\mathsf{ref}\, c$.

Lemma 6 (Closedness). *If* $\Sigma; \Delta^{+1}; \Gamma^{+1} \vdash \circ w^0 : [o]^0$, *then* $\mathrm{FV}(w^0) = \emptyset$.

Proof. By induction on the derivation of $\Sigma; \Delta^{+1}; \Gamma^{+1} \vdash \circ w^0 : [o]^0$.

Evaluation of $\mathsf{run}\, e$ at level 0 requires to view a value v at level 1 as a term to be evaluated at level 0. The following lemma says that this confusion in the levels is compatible with the type system.

Lemma 7 (Demotion). $\Sigma; \Delta^{+1}; \Gamma^{+1} \vdash v^{n+1} : t^{n+1}$ *implies* $\Sigma; \Delta; \Gamma \vdash v^{n+1} : t^n$.

Proof. By induction on the derivation of $\Sigma; \Delta^{+1}; \Gamma^{+1} \vdash v^{n+1} : t^{n+1}$.

The *reflective* nature of $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$ is fully captured by the Demotion Lemma and the following Promotion Lemma (which is not relevant to the proof of Type Safety).

Lemma 8. $\Sigma; \Delta; \Gamma \vdash e : t^n$ *implies* $e \in \mathsf{V}^{n+1}$ *and* $\Sigma; \Delta^{+1}; \Gamma^{+1} \vdash e : t^{n+1}$.

Finally, we establish the key result relating the type system to the operational semantics. This result entails that evaluation of a well-typed program $\emptyset; \emptyset \vdash e : t^0$ cannot raise an error, i.e. $\emptyset, e \overset{0}{\hookrightarrow} \mathsf{err}$ is not derivable.

Theorem 2 (Safety). $\mu_0, e \overset{n}{\hookrightarrow} d$ *and* $\Sigma_0 \models \mu_0$ *and* $\Sigma_0; \Delta^{+1}; \Gamma^{+1} \vdash e : t^n$ *imply that there exist* μ_1 *and* v^n *and* Σ_1 *such that* $d \equiv (\mu_1, v^n)$ *and* $\Sigma_0, \Sigma_1 \models \mu_1$ *and* $\Sigma_0, \Sigma_1; \Delta^{+1}; \Gamma^{+1} \vdash v^n : t^n$.

Proof. By induction on the derivation of the evaluation judgement $\mu_0, e \overset{n}{\hookrightarrow} d$.

4 Conservative Extension Result

This section shows that $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$ is a *conservative extension* of $\mathsf{MiniML}_{\mathsf{ref}}$ w.r.t. typing and operational semantics. When we need to distinguish the syntactic categories of $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$ from those of $\mathsf{MiniML}_{\mathsf{ref}}$ we use a superscript $\_^{\mathsf{meta}}$ for the formers, e.g. $\mathsf{E}^{\mathsf{meta}}$ denotes the set of $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$ terms, while E denotes the set of $\mathsf{MiniML}_{\mathsf{ref}}$ terms. We have the following inclusions between the syntactic categories of the two languages:

Lemma 9. $\mathsf{T} \subseteq \mathsf{C}^{\mathsf{meta}}$ *and* $\mathsf{E} \subseteq \mathsf{E}^{\mathsf{meta}}$ *and* $\mathsf{V} \subseteq \mathsf{V}^{0\,\mathsf{meta}}$.

Proof. Easy induction on the structure of $t \in \mathsf{T}$, $e \in \mathsf{E}$ and $v \in \mathsf{V}$.

There are minor mismatches between the typing and evaluation judgements of the two languages, thus we introduce three derived predicates, which simplify the formulation of the conservative extension result:

- $e\colon t$, i.e. e is a program of type t;
- $e \Downarrow$, i.e. evaluation of e may lead to a value;
- $e \Downarrow \mathsf{err}$, i.e. evaluation of e may lead to a run-time error.

The following table defines the three predicates in $\mathsf{MiniML}_{\mathsf{ref}}$ and $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$:

predicate	meaning in $\mathsf{MiniML}_{\mathsf{ref}}$	meaning in $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$
$e\colon t$	$\emptyset; \emptyset \vdash e\colon t$	$\emptyset; \emptyset; \emptyset \vdash e\colon t^0$
$e \Downarrow$	$\exists \mu, v.\ \emptyset, e \hookrightarrow \mu, v$	$\exists \mu, v.\ \emptyset, e \overset{0}{\hookrightarrow} \mu, v$
$e \Downarrow \mathsf{err}$	$\emptyset, e \hookrightarrow \mathsf{err}$	$\emptyset, e \overset{0}{\hookrightarrow} \mathsf{err}$

The conservative extension result can be stated as follows (the rest of the section establishes several facts, which combined together imply the desired result)

Theorem 3 (Conservative Extension). *$\mathsf{MiniML}_{\mathsf{ref}}$ and $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$ agree on the validity of the assertions $e\colon t$, $e \Downarrow$ and $e \Downarrow \mathsf{err}$, whenever $e \in \mathsf{E}$ and $t \in \mathsf{T}$.*

A typing judgement $\Sigma; \Gamma \vdash e\colon t$ for $\mathsf{MiniML}_{\mathsf{ref}}$ it is not appropriate for $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$, because $\Gamma\colon \mathsf{X} \overset{fin}{\to} \mathsf{T}$ and e lack the level information. Therefore, we introduce the following operation to turn a type assignment into a type-and-level assignment

$$\{x_i\colon t_i | i \in m\}^n \overset{\Delta}{=} \{x_i\colon t_i^n | i \in m\}$$

i.e. Γ^n assigns level n to all variables declared in Γ.

Proposition 1. *$\Sigma; \Gamma \vdash e\colon t$ in $\mathsf{MiniML}_{\mathsf{ref}}$ implies $\Sigma; \emptyset; \Gamma^0 \vdash e\colon t^0$ in $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$.*

Proof. Easy induction on the derivation of $\Sigma; \Gamma \vdash e\colon t$.

An immediate consequence of Proposition 1 is that $e\colon t$ in $\mathsf{MiniML}_{\mathsf{ref}}$ implies $e\colon t$ in $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$. For the converse, we need a translation from $\mathsf{T}^{\mathsf{meta}}$ to T.

Definition 2. *The function $\|\_\|$ from $\mathsf{T}^{\mathsf{meta}}$ to T is defined as*

$$\|[o]\| \overset{\Delta}{=} \|o\|$$
$$\|\langle t \rangle\| \overset{\Delta}{=} \|t\|$$

and it commutes with all other type-constructs of $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$. The extension to signatures Σ is point-wise; $\|\Gamma\|(x) = \|t\|$ when $\Gamma(x) = t^n$ and similarly for Δ.

Proposition 2. *$\Sigma; \Delta; \Gamma \vdash e\colon t^n$ implies $\|\Sigma\|; \|\Delta\|; \|\Gamma\| \vdash e\colon \|t\|$, provided $e \in \mathsf{E}$.*

Proof. By induction on the derivation of $\Sigma; \Delta; \Gamma \vdash e\colon t^n$.

The operational semantics of $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ may introduce Bullet (e.g. when manipulating the store), even when the evaluation starts in a configuration (μ, e) without occurrences of $\bullet$. Therefore, to relate the operational semantics of $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ and $\mathsf{MiniML}_{\mathsf{ref}}$, we introduce a partial function on $\mathsf{E}^{\mathsf{meta}}$ which erases Bullet from $\bullet e$ when $\mathrm{FV}(e) = \emptyset$.

Definition 3 (Erasure). *The partial function* $|\_|$ *on* $\mathsf{E}^{\mathsf{meta}}$ *is defined as*

$$|\bullet e| \stackrel{\Delta}{\equiv} \begin{cases} |e| & \text{if } \mathrm{FV}(e) = \emptyset \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

and it commutes with all other term-constructs of $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$.

Lemma 10. *The erasure enjoys the following properties:*

- *If* $\Sigma; \Delta; \Gamma \vdash e\colon t^n$ *and* $|e|$ *is defined, then* $\Sigma; \Delta; \Gamma \vdash |e|\colon t^n$;
- *if* $|e_2| \equiv e'_2$ *and* $|e_1| \equiv e'_1$ *then* $|e_2[x\colon= e_1]| \equiv e'_2[x\colon= e'_1]$.

Proof. The first part is by induction on the derivation of $\Sigma; \Delta; \Gamma \vdash e\colon t^n$; the second is by induction on the structure of e_2.

Definition 4 (Bisimulation). *The relation* $\mathsf{R} \subseteq \mathsf{E}^{\mathsf{meta}} \times \mathsf{E}_0$ *is given by* $e \mathrel{\mathsf{R}} e' \stackrel{\Delta}{\Longleftrightarrow} FV(e) = \emptyset$ *and* $|e| \equiv e'$.

The relation is extended to stores μ *and configurations* d *as follows:*
$\mu \mathrel{\mathsf{R}} \mu' \stackrel{\Delta}{\Longleftrightarrow} dom(\mu) = dom(\mu')$ *and* $\mu(l) \mathrel{\mathsf{R}} \mu'(l)$ *when* $l \in dom(\mu)$;
$d \mathrel{\mathsf{R}} d' \stackrel{\Delta}{\Longleftrightarrow} d = \mathsf{err} = d'$ *or* $(d = (\mu, e)$ *and* $d = (\mu', e')$ *where* $\mu \mathrel{\mathsf{R}} \mu'$ *and* $e \mathrel{\mathsf{R}} e')$.

The following proposition says that R is a bisimulation between the operational semantics of $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ and $\mathsf{MiniML}_{\mathsf{ref}}$.

Proposition 3. *If* $\mu \mathrel{\mathsf{R}} \mu'$ *and* $e \mathrel{\mathsf{R}} e'$, *then*

1. $\mu, e \stackrel{0}{\hookrightarrow} d$ *implies there exist (unique)* d' *such that* $d \mathrel{\mathsf{R}} d'$ *and* $\mu', e' \hookrightarrow d'$;
2. $\mu', e' \hookrightarrow d'$ *implies there exist* d *such that* $d \mathrel{\mathsf{R}} d'$ *and* $\mu, e \stackrel{0}{\hookrightarrow} d$.

Proof. The first part is by induction on the derivation of $\mu, e \stackrel{0}{\hookrightarrow} d$. The second part is by lexicographic induction on the derivation of $\mu', e' \hookrightarrow d'$ and the number of top-level Bullets in e (i.e. n such that $e \equiv \bullet^n w$).

This implies the conservative extension result for the predicates $e \Downarrow$ and $e \Downarrow \mathsf{err}$.

5 Conclusions and Further Research

In this section we discuss possible improvements to the type system and variations to the syntax and operational semantics of $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$.

Sub-typing. In $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ sub-typing arises naturally, e.g. one expects $[o] \leq o$ for any open type $o \in \mathsf{O}$.

Before adding a sub-sumption rule $\dfrac{\Sigma; \Delta; \Gamma \vdash e: t_1^n}{\Sigma; \Delta; \Gamma \vdash e: t_2^n}\ t_1 \leq t_2$ to $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$, it is better to adopt a more general syntax for types t and closed types c

$$
\begin{array}{l}
t \in \mathsf{T} ::= \mathsf{nat} \mid t_1 \rightarrow t_2 \mid \mathsf{ref}\ c \mid \langle t \rangle \mid [t] \\
c \in \mathsf{C} ::= \mathsf{nat} \mid t_1 \rightarrow c_2 \mid \mathsf{ref}\ c \mid [t]
\end{array}
$$

and let the sub-typing rule derive $[c] = c$. One expects the usual sub-typing rules for functional and references types, and it seems natural to require the Code and Closed type constructors to be covariant, i.e.

$$
\frac{t_1' \leq t_1 \quad t_2 \leq t_2'}{t_1 \rightarrow t_2 \leq t_1' \rightarrow t_2'} \quad \frac{c' \leq c \quad c \leq c'}{\mathsf{ref}\ c \leq \mathsf{ref}\ c'} \quad \frac{t \leq t'}{\langle t \rangle \leq \langle t' \rangle} \quad \frac{t \leq t'}{[t] \leq [t']}
$$

while sub-typing axioms, which generate non trivial relations, are

$$
[t] \leq t \qquad c \leq [c] \qquad [t_1 \rightarrow t_2] \leq [t_1] \rightarrow [t_2] \qquad [\langle t \rangle] \leq \langle [t] \rangle
$$

From the sub-typing axioms and rules above one can derive the following facts:

- $t \leq c$ implies $t \in \mathsf{C}$, by induction on the derivation of $t \leq c$;
- $[c] = c$ and $c \rightarrow [t] = [c \rightarrow t]$, while the following sub-typing are strict $[\langle t \rangle] < \langle t \rangle$ and $[\langle t_1 \rangle] \rightarrow [\langle t_2 \rangle] < [\langle t_1 \rangle \rightarrow \langle t_2 \rangle]$.

We plan to investigate the addition of sub-typing and its effects on type safety.

Useless-variable annotation. The binder $\bullet e$ of $\mathsf{MiniML}^{\mathsf{meta}}_{\mathsf{ref}}$ takes an all or nothing approach. One could provide a more fine-grained annotation $(x)e$, which allows to name a useless variable. The typing rules for $(x)e$ are the obvious one:

$$
\frac{\Sigma; \Delta; \Gamma, x: t^m \vdash e: c^n}{\Sigma; \Delta; \Gamma \vdash (x)e: c^n}\ m > n \qquad \frac{\Sigma; \Delta, x: t^m; \Gamma \vdash e: c^n}{\Sigma; \Delta; \Gamma \vdash (x)e: c^n}\ m > n
$$

One can define the derived notation $(X)e$, where X is a finite set/sequence of variables, by induction on the cardinality of X: $(\emptyset)e \stackrel{\Delta}{\equiv} e$, $(x, X)e \stackrel{\Delta}{\equiv} (x)(X)e$. One might identify $\bullet e$ with $\tilde{e} \stackrel{\Delta}{\equiv} (X)e$, where $X = \mathrm{FV}(e)$. However, at the operational level such identification is not right. In fact, the rule

$$
\frac{\mu_0, e_1 \stackrel{0}{\hookrightarrow} \mu_1, \bullet\lambda x.e \quad \mu_1, e_2 \stackrel{0}{\hookrightarrow} \mu_2, v_2 \quad \mu_2, \bullet(e[x := v_2]) \stackrel{0}{\hookrightarrow} \mu_3, v}{\mu_0, e_1\ e_2 \stackrel{0}{\hookrightarrow} \mu_3, v}
$$

is not an instance of

$$
\frac{\mu_0, e_1 \stackrel{0}{\hookrightarrow} \mu_1, (X)\lambda x.e \quad \mu_1, e_2 \stackrel{0}{\hookrightarrow} \mu_2, v_2 \quad \mu_2, ((X)e)[x := v_2] \stackrel{0}{\hookrightarrow} \mu_3, v}{\mu_0, e_1\ e_2 \stackrel{0}{\hookrightarrow} \mu_3, v}
$$

since the free variables in v_2 are bound by $\bullet_{\_}$, but not by $(X)_{\_}$. This seems to suggest that one might want to maintain $\bullet e$ even in the presence of $(x)e$. On the other hand, the conservative extension of $\mathsf{MiniML}_{\mathsf{ref}}$ into $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$ seems to become simpler if we use $(x)e$ (and a suitable adaptation of the operational semantics) instead of $\bullet e$.

Acknowledgements

We would like to thank the anonymous referees for their valuable comments (any failure to fully exploit them is our fault). This paper would not have been conceived without the previous work in collaboration with Zino Benaissa, Tim Sheard and Walid Taha, who have introduced us to the challenges of multi-stage programming. Finally, we would like to thank Tim Sheard for his stimulating criticisms on previous attempts, and Walid Taha for many discussions.

References

[B97] Zine El-Abidine Benaissa Explicit Substitution Calculi as a Foundation of Functional Programming Languages Implementations Phd thesis, INRIA, 1997.

[BMTS99] Zine El-Abidine Benaissa, Eugenio Moggi, Walid Taha, and Tim Sheard. Logical modalities and multi-stage programming. In *Federated Logic Conference (FLoC) Satellite Workshop on Intuitionistic Modal Logics and Applications (IMLA)*, July 1999.

[CDDK86] Dominique Clement, Joelle Despeyroux, Thierry Despeyroux, and Gilles Kahn. A simple applicative language: Mini-ML. In *Proceedings of the 1986 ACM Conference on Lisp and Functional Programming*, pages 13–27. ACM, ACM, August 1986.

[CMT00] Cristiano Calcagno, Eugenio Moggi, Walid Taha. Closed Types as a Simple Approach to Safe Imperative Multi-Stage Programming. In *Proceedings of ICALP 2000*, volume 1853 of LNCS, pages 25–36, Springer, 2000.

[Dav96] Rowan Davies. A temporal-logic approach to binding-time analysis. In *Proceedings, 11$^{\text{th}}$ Annual IEEE Symposium on Logic in Computer Science*, pages 184–195, New Brunswick, July 1996. IEEE Computer Society Press.

[DP96] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. In *23rd Annual ACM Symposium on Principles of Programming Languages (POPL'96)*, pages 258–270, St. Petersburg Beach, January 1996.

[LM99] Jean-Jacques Levy and Luc Maranget. Explicit Substitutions and Programming Languages. In *19th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, Chennai, India, December 1999.

[Met00] The MetaML Home Page, 2000. Provides source code and documentation online at http://www.cse.ogi.edu/PacSoft/projects/metaml/.

[MTBS99] Eugenio Moggi, Walid Taha, Zine El-Abidine Benaissa, and Tim Sheard. An idealized MetaML: Simpler, and more expressive. In *European Symposium on Programming (ESOP)*, volume 1576 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag, 1999.

[Tah99] Walid Taha. *Multi-Stage Programming: Its Theory and Applications.* PhD thesis, Oregon Graduate Institute of Science and Technology, July 1999.

[TBS98] Walid Taha, Zine-El-Abidine Benaissa, and Tim Sheard. Multi-stage programming: Axiomatization and type-safety. In *25th International Colloquium on Automata, Languages, and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 918–929, Aalborg, July 1998.

[TS97] Walid Taha and Tim Sheard. Multi-stage programming with explicit annotations. In *Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and semantic based program manipulations PEPM'97, Amsterdam*, pages 203–217. ACM, 1997.

[TS00] Walid Taha and Tim Sheard. MetaML: Multi-stage programming with explicit annotations. *Theoretical Computer Science*, 248(1-2), 2000.

[WS99] Mitchell Wand, Igor Siveroni. Constraint Systems for Useless Variable Elimination. In *Proceedings of 26th ACM Symposium on Principles of Programming Languages (POPL)*, pages 291–302, 1999.

[WF94] Andrew K. Wright and Matthias Felleisen. A Syntactic Approach to Type Soundness. *Information and Computation*, 115(1):38–94, 1994.

$$\text{(case*)}\ \frac{\Sigma;\Delta;\Gamma\vdash e\colon \mathsf{nat}^n \quad \Sigma;\Delta;\Gamma\vdash e_1\colon t^n \quad \Sigma;\Delta,x\colon\mathsf{nat}^n;\Gamma\vdash e_2\colon t^n}{\Sigma;\Delta;\Gamma\vdash(\mathsf{case}\,e\,\mathsf{of}\,\mathsf{z}\to e_1\mid \mathsf{s}\,x\to e_2)\colon t^n}$$

$$\frac{\Sigma;\Delta^{\leq n};\emptyset\vdash e\colon t^n}{\Sigma;\Delta;\Gamma\vdash[e]\colon[t]^n} \qquad \frac{\Sigma;\Delta;\Gamma\vdash e_1\colon[t_1]^n \quad \Sigma;\Delta,x\colon t_1^n;\Gamma\vdash e_2\colon t_2^n}{\Sigma;\Delta;\Gamma\vdash(\mathsf{let}\,[x]=e_1\,\mathsf{in}\,e_2)\colon t_2^n}$$

$$\text{(fix*)}\ \frac{\Sigma;\Delta^{\leq n},x\colon t^n;\emptyset\vdash e\colon t^n}{\Sigma;\Delta;\Gamma\vdash\mathsf{fix}\,x.e\colon t^n} \qquad \text{(close*)}\ \frac{\Sigma;\Delta;\Gamma\vdash e\colon c^n}{\Sigma;\Delta;\Gamma\vdash[e]\colon[c]^n}$$

Fig. 5. Type System for $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{BN}}$

A $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{BN}}$

This section recalls the syntax and type system of $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{BN}}$, to help in a comparison with $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$. The types t and closed types c of $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{BN}}$ are defined as

$$t\in\mathsf{T}::=c\mid t_1\to t_2\mid\langle t\rangle \qquad c\in\mathsf{C}::=\mathsf{nat}\mid[t]\mid\mathsf{ref}\,c$$

Remark 3. Function types are never closed, the types $[c]$ and c are not identified.

The set of $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{BN}}$ terms is defined as

$$\begin{aligned} e\in\mathsf{E}::=\ & x\mid\lambda x.e\mid e_1\ e_2\mid\mathsf{fix}\,x.e\mid\mathsf{z}\mid\mathsf{s}\,e\mid(\mathsf{case}\,e\,\mathsf{of}\,\mathsf{z}\to e_1\mid\mathsf{s}\,x\to e_2)\mid\\ & \langle e\rangle\mid\tilde{\ }e\mid\mathsf{run}\ e\mid[e]\mid(\mathsf{let}\,[x]=e_1\,\mathsf{in}\,e_2)\mid\\ & \mathsf{ref}\,e\mid\,!\,e\mid e_1\colon=e_2\mid l\mid\mathsf{fault}\end{aligned}$$

Remark 4. The constant fault leads to a run-time error when evaluated at level 0, and evaluates to itself at higher levels. Operationally, fault is equivalent to the $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$ term $\bullet x$. There is an explicit closed construct $[e]$, and one let-binder $(\mathsf{let}\,[x]=e_1\,\mathsf{in}\,e_2)$.

Figure 5 summarizes the typing rules of $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{BN}}$ which differ from those of $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$. The main differences are:

- (case*) corresponds to declare the bound variable in Δ, instead of Γ, and is only used to simplify the translation of $\mathsf{MiniML}_{\mathsf{ref}}$ in $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{BN}}$.
- (close*) is necessary because there is no identification of $[c]$ with c.
- (fix*) can type recursive definitions (e.g. of closed functions) that are not typable with (fix). For instance, from $\emptyset;f'\colon[t_1\to t_2]^n,x\colon t_1^n\vdash e\colon t_2^n$ one cannot derive $\mathsf{fix}\,f'.[\lambda x.e]\colon[t_1\to t_2]^n$, while the following modified term $\mathsf{fix}\,f'.(\mathsf{let}\ [f]=f'\ \mathsf{in}\ [\lambda x.e[f'\colon=[f]]])$ has the right type, but the wrong behavior (it diverges!). The rule (fix*) allows to type $[\mathsf{fix}\,f.\lambda x.e[f'\colon=[f]]]$, which has the desired operational behavior.
  In $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{meta}}$ the (fix*) rule is not necessary: assuming a unit type $()$, one could write the term $\mathsf{fix}\,f'.(\mathsf{let}_c f=\lambda().f'\,\mathsf{in}\,\lambda x.e[f'\colon=f()])$ which has the desired type and does not diverge; this term is not typable in $\mathsf{MiniML}_{\mathsf{ref}}^{\mathsf{BN}}$ because $f\colon()\to(t_1\to t_2)$ would not have a closed type.