Metalanguages and Applications

Eugenio Moggi

DISI, Univ. of Genova
via Dodecaneso 35
16146 Genova, Italy
moggi@disi.unige.it

- AIM: exemplify use of metalanguages for *describing* programming languages
- overview of notes:
  - sec 3: background on logical frameworks
  - sec 4: main ideas
  - sec 5: refinement of main ideas
- abstract syntax, translations and LF (sec 3)
- computational types and monadic approach (sec 4.2)
- incremental approach (sec 4.3)

? recursive definitions: cpos and axiomatization in LF

? computational types revised (sec 5.5)

Programming languages and semantics

Rigorous approaches:

- operational: abstract machine
- denotational: math. model, Tarski's semantics
- axiomatic/algebraic: inference rules (for equivalence)

Trade-offs

desiderata	op. sem.	den. sem.	ax. sem.
simplicity			
consistency			
levels of abstraction			
modifiability			
validation tool			

Good-fit criteria

- observations and computational adequacy
- soundness (completeness) of
  - inference rules w.r.t. model
  - model w.r.t. observational equivalence

Metalanguages and denotational semantics

- denotational semantics

$$PL \xrightarrow{interp} \mathcal{C}$$

in a category: **Set**, **Cpo**, functor category, topos,...

- semantics via translation

$$PL \xrightarrow{transl} ML \xrightarrow{interp} \mathcal{C}$$

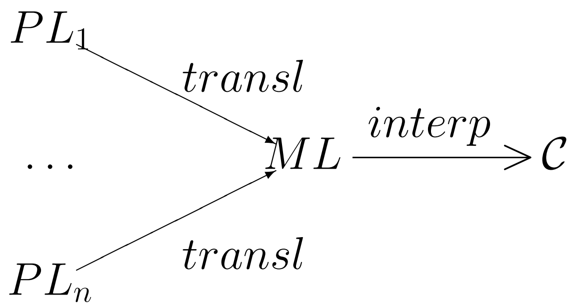
in a **typed** metalanguage

Approach to den. sem. of PL: desiderata

- simplicity/can hide mathematical complexities
- scalability of approach to complex PL
- modularity/first work in isolation then combine
- reuse know-how accumulated in den. semantics

Semantic via translation: advantages

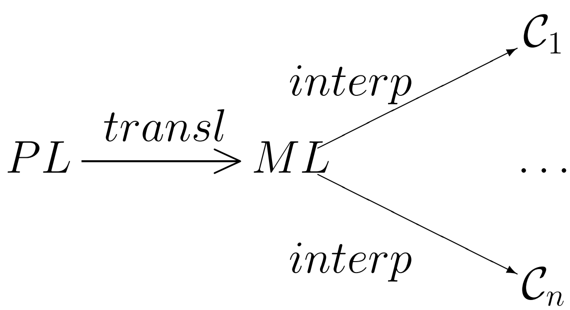
- **reuse** same ML for several PLs



  - translation simpler than direct interpretation!?
  - transfer of properties/results from ML to PL: reasoning principles, computational adequacy

- **choice** of ML to meet certain criteria:
  - ML based on **few orthogonal** concepts
  - ML equipped with a **logic** used as specification language or for formalizing reasoning principles
  - ML as *internal language* for a class of categories

- **hiding** semantic categories under ML



Choice of ML: criteria

ML built on top of a fairly standard typed λ-calculus, more controversial issues:

- should ML be equipped with some logic (which)?

  YES, to abstract reasoning principles from semantics!

- should ML be a PL (i.e. have op. sem.)?

  NO, may need choice of op. semantics (CBV or CBN?), restriction on types (no dependent types!), *non-standard* equational axioms.

- what is gained by giving semantics to (complex) PL via a ML, rather than directly?

  nothing, if translation of PL into ML or semantics of ML are as complex as direct semantics of PL.

  – **monadic approach** for structuring translation from PL to ML with **auxiliary notation**

  $$PL \xrightarrow{transl} ML(\Sigma) \xrightarrow{transl} ML$$

  – **incremental** definition of auxiliary notation

  $$PL \xrightarrow{transl} ML(\Sigma_n) \xrightarrow{transl} \ldots$$

  $$\xrightarrow{transl} ML(\Sigma_0) \xrightarrow{transl} ML$$

LF Judgements

- $\Gamma \vdash$, set I
- $\Gamma \vdash A\colon Type_i$, family of sets $\langle X_i | i \in I \rangle$
- $\Gamma \vdash M\colon A$, family of elements $\langle x_i \in X_i | i \in I \rangle$

Pseudo-terms

$$A, M \in Exp ::= \ x \mid Type_i \mid \Pi x\colon A_1.A_2 \mid \lambda x\colon A.M \mid M_1 M_2$$

Formation rules

$$\text{empty}\frac{}{\emptyset \vdash}$$

$$\text{ext}\frac{\Gamma \vdash A\colon Type_i}{\Gamma, x\colon A \vdash}\ x \notin \mathrm{DV}(\Gamma)$$

$$\text{type-}\in\frac{\Gamma \vdash}{\Gamma \vdash Type_i\colon Type_{i+1}}\ i \geq 0$$

$$\text{type-}\subset\frac{\Gamma \vdash A\colon Type_i}{\Gamma \vdash A\colon Type_j}\ i < j$$

$$\text{var}\frac{\Gamma \vdash}{\Gamma \vdash x\colon A}\ A = \Gamma(x)$$

$$\Pi\frac{\Gamma \vdash A_1\colon Type_i \qquad \Gamma, x\colon A_1 \vdash A_2\colon Type_i}{\Gamma \vdash (\Pi x\colon A_1.A_2)\colon Type_i}$$

$$\lambda\frac{\Gamma \vdash A_1\colon Type_i \qquad \Gamma, x\colon A_1 \vdash M_2\colon A_2}{\Gamma \vdash (\lambda x\colon A_1.M_2)\colon (\Pi x\colon A_1.A_2)}$$

$$\text{app}\frac{\Gamma \vdash M\colon (\Pi x\colon A_1.A_2) \qquad \Gamma \vdash M_1\colon A_1}{\Gamma \vdash M M_1\colon A_2[x := M_1]}$$

$$\text{conv}\frac{\Gamma \vdash M\colon A_1 \qquad \Gamma \vdash A_2\colon Type_i}{\Gamma \vdash M\colon A_2}\ A_1 =_{\beta\eta} A_2$$

$=_{\beta\eta}$ is $\beta\eta$-conversion on pseudo-terms.

Derived notions

- Σ LF-signature: $\Sigma \vdash$
- $\Gamma \vdash_\Sigma J$ relativized judgement: $\Sigma, \Gamma \vdash J$

  $L(\Sigma)$=set of derivable judgements $\Gamma \vdash_\Sigma J$
- $I\colon \Sigma' \to \Sigma$ realization of Σ' in Σ:
  - $\emptyset\colon \emptyset \to \Sigma$ realization
  - $(I, x{:=}M)\colon (\Sigma', x\colon A) \to \Sigma$ realization iff

    $I\colon \Sigma' \to \Sigma$ realization and $\vdash_\Sigma M\colon A[I]$ derivable

    $A[I]$ parallel substitution of I in A

Concrete syntax for PL_{fun}

types $\tau \in T ::= \ bool \mid \tau_1 \Rightarrow \tau_2$

identifiers $x \in Id ::=$ an infinite set

expressions $e \in Exp ::= \ x \mid \bot \mid$

$tt \mid ff \mid if(e, e_1, e_2) \mid$

$(\lambda x{:}\,\tau_1.e_2) \mid ap(e, e_1)$

CBV operational semantics $\Downarrow \subset Exp \times Val$

$v \in Val ::= \ x \mid tt \mid ff \mid (\lambda x{:}\,\tau_1.e_2)$

$$\text{val}\,\frac{}{v \Downarrow v}$$

$$if\,\frac{e \Downarrow tt \quad e_1 \Downarrow v}{if(e, e_1, e_2) \Downarrow v} \qquad if\,\frac{e \Downarrow ff \quad e_2 \Downarrow v}{if(e, e_1, e_2) \Downarrow v}$$

$$ap\,\frac{e \Downarrow (\lambda x{:}\,\tau_1.e_2) \quad e_1 \Downarrow v_1 \quad e_2[x{:=}\,v_1] \Downarrow v}{ap(e, e_1) \Downarrow v}$$

CBN as modification of CBV

$v \in Val ::= \ tt \mid ff \mid (\lambda x{:}\,\tau_1.e_2)$

$$ap\,\frac{e \Downarrow (\lambda x{:}\,\tau_1.e_2) \quad e_2[x{:=}\,e_1] \Downarrow v}{ap(e, e_1) \Downarrow v}$$

Typing rules for PL_{fun}

$$var \frac{}{\Gamma \vdash x\colon Exp[\tau]}\ \Gamma(x) = Id[\tau]$$

$$tt \frac{}{\Gamma \vdash tt\colon Exp[bool]} \qquad ff \frac{}{\Gamma \vdash ff\colon Exp[bool]}$$

$$if \frac{\Gamma \vdash e\colon Exp[bool] \quad \Gamma \vdash e_1, e_2\colon Exp[\tau]}{\Gamma \vdash if(e, e_1, e_2)\colon Exp[\tau]} \qquad \bot \frac{}{\Gamma \vdash \bot\colon Exp[\tau]}$$

$$ab \frac{\Gamma; x\colon Id[\tau_1] \vdash e_2\colon Exp[\tau_2]}{\Gamma \vdash \lambda x\colon \tau_1.e_2\colon Exp[\tau_1 \Rightarrow \tau_2]} \qquad ap \frac{\Gamma \vdash e\colon Exp[\tau_1 \Rightarrow \tau_2] \quad \Gamma \vdash e_1\colon Exp[\tau_1]}{\Gamma \vdash ap(e, e_1)\colon Exp[\tau_2]}$$

LF-signature Σ_{fun}

types	$T\colon Type$
	$bool\colon T$
	$\Rightarrow\colon T, T \to T$
ident	$Id\colon T \to Type$
expr	$Exp\colon T \to Type$
	$var\colon \Pi X\colon T.Id(X) \to Exp(X)$
	$\bot\colon \Pi X\colon T.Exp(X)$
bool	$tt, ff\colon Exp(bool)$
	$if\colon \Pi X\colon T.Exp(bool), Exp(X), Exp(X) \to Exp(X)$
$\Rightarrow$	$ab\colon \Pi X_1, X_2\colon T.(Id(X_1) \to Exp(X_2)) \to Exp(X_1 \Rightarrow X_2)$
	$ap\colon \Pi X_1, X_2\colon T.Exp(X_1 \Rightarrow X_2), Exp(X_1) \to Exp(X_2)$

LF-signature Σ_{ML}

$$\Sigma_\times \quad \frac{\text{unit} \quad 1\colon Type}{\ast\colon 1}$$

$$\Sigma_+ \quad \frac{\text{sum} \quad +\colon Type, Type \to Type}{\begin{array}{ll} \text{inject} & in_i\colon \Pi X_1, X_2\colon Type. X_i \to (X_1 + X_2) \\ \text{case} & case\colon \Pi X_1, X_2, X\colon Type. \\ & \quad (X_1 \to X), (X_2 \to X) \to (X_1 + X_2) \to X \end{array}}$$

write (case M of $x_1.M_1|x_2.M_2$) for
$case(\tau_1, \tau_2, \tau, (\lambda x_1\colon \tau_1.M_1), (\lambda x_2\colon \tau_2.M_2), M)$

CBV translation $I\colon \Sigma_{fun} \to \Sigma_{ML}$

$$\begin{array}{rl}
T^*\colon= & Type \\ \hline
bool^*\colon= & 1 + 1 \\
\Rightarrow^*(X_1, X_2)\colon= & X_1 \to (X_2 + 1) \\
Id^*(X)\colon= & X \\
Exp^*(X)\colon= & X + 1 \\ \hline
var^*(X, x)\colon= & in_1(x) \\
tt^*\colon= & in_1(in_1(\ast)) \\
ff^*\colon= & in_1(in_2(\ast)) \\
if^*(X, c, c_1, c_2)\colon= & \text{case } c \text{ of} \\
 & x.\ \text{case } x \text{ of } c_1|c_2 \\
 & in_2(\ast) \\
ab^*(X_1, X_2, f)\colon= & in_1(f) \\
ap^*(X_1, X_2, c, c_1)\colon= & \text{case } c \text{ of} \\
 & f.\ \text{case } c_1 \text{ of } x.f(x)|in_2(\ast) \\
 & in_2(\ast) \\
\bot^*\colon= & in_2(\ast)
\end{array}$$

LF-signature Σ_{ML}

Σ_{eq}	props	$Prop\colon Type$
	proofs	$pr\colon Prop \to Type$
	equal	$eq\colon \Pi X\colon Type. X, X \to Prop$
	refl	$\Pi X\colon Type. \Pi x\colon X. x = x$
	subst	$\Pi X\colon Type, P\colon X \to Prop, x, y\colon X. x = y, P(x) \to P(y)$
Σ_{ext}	Π-ext	$\Pi X\colon Type, F\colon X \to Type, f, g\colon (\Pi x\colon X. Fx). (\Pi x\colon X. fx = gx) \to f = g$
$\Sigma_{\times}$	unit	$1\colon Type$
	product	$\times\colon Type, Type \to Type$
		$*\colon 1$
	pairing	$pair\colon \Pi X_1, X_2\colon Type. X_1, X_2 \to X_1 \times X_2$
	project	$\pi_i\colon \Pi X_1, X_2\colon Type. (X_1 \times X_2) \to X_i$
		$\Pi x\colon 1. x = *$
		$\Pi X_1, X_2\colon Type, x_1\colon X_1, x_2\colon X_2. \pi_i(\langle x_1, x_2 \rangle) = x_i$
		$\Pi X_1, X_2\colon Type, x\colon X_1 \times X_2. \langle \pi_1(x), \pi_2(x) \rangle = x$
Σ_{+}	empty	$0\colon Type$
	sum	$+\colon Type, Type \to Type$
		$0\colon \Pi X\colon Type. 0 \to X$
	inject	$in_i\colon \Pi X_1, X_2\colon Type. X_i \to (X_1 + X_2)$
	case	$case\colon \Pi X_1, X_2, X\colon Type. (X_1 \to X), (X_2 \to X) \to (X_1 + X_2) \to X$
		$\Pi X\colon Type, x\colon 0, y\colon X. 0(X, x) = y$
		$\Pi X_1, X_2\colon Type, f_1\colon (X_1 \to X), f_2\colon (X_2 \to X), x\colon X_i. case(f_1, f_2, in_i(x)) = f_i(x)$
		$\Pi X_1, X_2, X\colon Type, f\colon (X_1 \times X_2) \to X. case(f \circ in_1, f \circ in_2) = f$

The scalability problem

- when a PL is extended, its op./den. semantics may need to be extensively redefined
- the problem remains when giving semantics via translation in a typed ML (as considered so far)
- [Mos90b] suggests the use of **auxiliary notation** to make semantic definitions more reusable.
- [Mog91] identifies *monads* as a structuring device for den. semantics (but not for op. semantics!).

Monadic approach: basic idea

- add to ML type constructor T, elements of $T\tau$ (computational type) as programs computing values in τ
- interpretation of T should fit *computational features* of PL, but some operations (to specify order of evaluation) common to all T
- use metalanguage $ML_T(\Sigma)$ with computational types and signature Σ of additional operations on T
- **monadic approach** to den. semantics, given PL:
  1. identify $ML_T(\Sigma)$, T and Σ like interface of ADT
  2. define translation of PL into $ML_T(\Sigma)$
  3. give model of $ML_T(\Sigma)$, via translation into ML.

  with a good Σ, translation $PL \to ML_T(\Sigma)$ is simple, and *no* need to redefine it when PL is extended.

Computational types

$$
\begin{array}{l|l}
\Sigma_T & T\colon Type \to Type \\
\hline
 & val^T\colon \Pi X\colon Type.X \to TX \\
 & let^T\colon \Pi X_1, X_2\colon Type.(X_1 \to TX_2), TX_1 \to TX_2 \\
\hline
 & \Pi X_1, X_2\colon Type, x\colon X_1, f\colon (X_1 \to TX_2). \\
 & \qquad let(f, val(x)) = f(x) \\
 & \Pi X\colon Type, c\colon TX. \\
 & \qquad let(val, c) = c \\
 & \Pi X_1, X_2, X_3\colon Type, c\colon TX_1, f\colon (X_1 \to TX_2), g\colon (X_2 \to TX_3). \\
 & \qquad let(g, let(f, c)) = let(let(g) \circ f, c)
\end{array}
$$

derived notation

- $[e]_T$ for $val^T(e)$
- $\mathrm{let}_T\, x{\Leftarrow}e_1\,\mathrm{in}\,e_2$ for $let^T(\lambda x\colon \tau_1.e_2, e_1)$
- $Tf\colon T\tau_1 \to T\tau_2$ for $\lambda c\colon T\tau_1.\mathrm{let}\, x{\Leftarrow}c\,\mathrm{in}\,[f(x)]$,
  where $f\colon \tau_1 \to \tau_2$
- $\mu\colon T^2\tau \to T\tau$ for $\lambda c\colon T^2\tau.\mathrm{let}\, x{\Leftarrow}c\,\mathrm{in}\, x$
- $\mathrm{let}\,\overline{x}{\Leftarrow}\overline{e}\,\mathrm{in}\,e$ for $\mathrm{let}\,x_1{\Leftarrow}e_1\,\mathrm{in}\,(\ldots(\mathrm{let}\,x_n{\Leftarrow}e_n\,\mathrm{in}\,e)\ldots)$
- $\langle \overline{x} \Leftarrow \overline{e}, e\rangle$ for $\mathrm{let}\,\overline{x}, x{\Leftarrow}\overline{e}, e\,\mathrm{in}\,[\langle \overline{x}, x\rangle]$
- $\mathrm{let}\,\langle\overline{x}\rangle{\Leftarrow}c\,\mathrm{in}\,e(x_1, \ldots, x_n)$ for $\mathrm{let}\,x{\Leftarrow}c\,\mathrm{in}\,e(\pi_1(x), \ldots, \pi_n(x))$.

Computational types: simple examples

- $TX = |T_{Th}(X)|$
  Th single sorted algebraic theory
  $T_{Th}(X)$ free Th-algebra over X
  - $TX = X$, terminating (functional) programs
  - $TX = X + \{\perp\}$, programs which may diverge
  - $TX = \mathcal{P}_{fin}(X)$, nondeterministic programs
  - $TX = (X + E)$, programs with exceptions
  - $TX = (\mu X'.X + X')$, parallel programs
- $TX = (X \times S)^S$, imperative programs
  $TX = X^S$, state-reading programs
- $TX = R^{(R^X)}$, programs with a continuation
- $TX = (X \times N)$, programs with timers

Variations and Combinations

- variations for nondeterministic programs:

  $TX = \mathcal{P}(X)$, subsets of X

  $TX = \mathcal{P}_\omega(X)$, countable subsets of X

- combinations imperative programs + exceptions:

  $TX = ((X + E) \times S)^S$

  $TX = ((X \times S) + E)^S$

- combination non-deterministic parallel programs:

  $TX = \mu X'.\mathcal{P}_{fin}(X + X')$

- variation parallel communicating programs:

  $TX = \mu X'.\mathcal{P}_{fin}(X + (A \times X'))$

  $T0$ =finite synchronization trees/strong bisimulation

- combination parallel imperative programs:

  $TX = \mu X'.\mathcal{P}_{fin}((X + X') \times S)^S$

Translation $PL \to ML_T(\Sigma)$: general pattern

$$\Sigma_{PL} \longrightarrow \Sigma_{ML} + \Sigma_T + \Sigma$$

CBV translation of Σ_{fun} in $ML_T(\Sigma)$

Σ divergence $\bot\colon \Pi X\colon Type.T(X)$

- $$\begin{array}{rl} T^*{:=} & Type \\ \hline bool^*{:=} & 1+1 \\ \Rightarrow^*(X_1,X_2){:=} & X_1 \to T(X_2) \\ Id^*(X){:=} & X \\ Exp^*(X){:=} & T(X) \\ \hline var^*(X,x){:=} & [x] \\ tt^*{:=} & [in_1(*)] \\ ff^*{:=} & [in_2(*)] \\ if^*(X,c,c_1,c_2){:=} & \mathrm{let}\ x{\Leftarrow}c\ \mathrm{in}\ (\mathrm{case}\ x\ \mathrm{of}\ c_1|c_2) \\ ab^*(X_1,X_2,f){:=} & [f] \\ ap^*(X_1,X_2,c,c_1){:=} & \mathrm{let}\ f,x{\Leftarrow}c,c_1\ \mathrm{in}\ f(x) \\ \bot^*{:=} & \bot \end{array}$$

CBN translation as modification of CBV

- $$\begin{array}{rl} \hline \Rightarrow^*(X_1,X_2){:=} & T(X_1) \to T(X_2) \\ Id^*(X){:=} & T(X) \\ \hline var^*(X,c){:=} & c \\ ap^*(X_1,X_2,c,c_1){:=} & \mathrm{let}\ f{\Leftarrow}c\ \mathrm{in}\ f(c_1) \end{array}$$

Extensions to PL_{fun}

PL_{imp} mutable store (add types *unit* and *nat*)

locations $l \in Loc ::=$ a set

expressions $e \in Exp ::= \ldots \mid l \mid l := e$

PL_{exc} exception handling

exceptions $n \in Exn ::=$ a set

expressions $e \in Exp ::= \ldots \mid rse(n) \mid hdl(n, e_1, e_2)$

PL_{nd} non-deterministic choice

expressions $e \in Exp ::= \ldots \mid or(e_1, e_2)$

PL_{par} parallelism

expressions $e \in Exp ::= \ldots \mid por(e_1, e_2) \mid pap(e_1, e_2)$

Corresponding LF-signature extensions

Σ_{imp} locations $Loc : Type$

$get : Loc \to Exp(nat)$

$set : Loc, Exp(nat) \to Exp(unit)$

Σ_{exc} exceptions $Exn : Type$

$rse : \Pi X : T. Exn \to Exp(X)$

$hdl : \Pi X : T. Exn, Exp(X), Exp(X) \to Exp(X)$

Σ_{nd} $or : \Pi X : T. Exp(X), Exp(X) \to Exp(X)$

Σ_{par} $por : \Pi X : T. Exp(X), Exp(X) \to Exp(X)$

$pap : \Pi X_1, X_2 : T. Exp(X_1 \Rightarrow X_2), Exp(X_1) \to Exp(X_2)$

Extensions of CBV translation of Σ_{fun}

$$\Sigma \quad \frac{\text{locations} \quad L\colon Type}{\begin{array}{ll}\text{lookup} & lkp\colon L \to TN \\ \text{update} & upd\colon L, N \to T1\end{array}}$$

- $$\frac{Loc^*\colon= \ L}{\begin{array}{rl} get^*(l)\colon= & lkp(l) \\ set^*(l,c)\colon= & \text{let } x \Leftarrow c \text{ in } upd(l,x)\end{array}}$$

$$\Sigma \quad \frac{\text{exceptions} \quad E\colon Type}{\begin{array}{ll}\text{test} & eq\colon E, E \to 1+1 \\ \text{raise} & rse\colon \Pi X\colon Type.E \to TX \\ \text{handle} & hdl\colon \Pi X\colon Type.(E \to TX), TX \to TX\end{array}}$$

- $$\frac{Exn^*\colon= \ E}{\begin{array}{rl} rse^*\colon= & rse \\ hdl^*(X,n,c_1,c_2)\colon= & hdl(X, (\lambda x\colon E.\text{case } eq(x,n) \text{ of } c_1 | rse(x)), c_2)\end{array}}$$

Σ choice $or\colon \Pi X\colon Type.TX, TX \to TX$

- $or^*\colon= \ or$

Modifications to CBV translation of Σ_{fun}

Σ one-step $\delta\colon \Pi X\colon Type.TX \to TX$

or-par $por\colon \Pi X\colon Type.TX, TX \to TX$

and-par $pand\colon \Pi X_1, X_2\colon Type.TX_1, TX_2 \to T(X_1 \times X_2)$

better signature (see 5.6), when fix is available,

or $or\colon \Pi X\colon Type.TX, TX \to TX$

one-step $\delta\colon \Pi X\colon Type.TX \to TX$

case-step $C\colon \Pi X_1, X_2\colon Type.$
$(X_1 \to TX_2), (TX_1 \to TX_2), TX_1 \to TX_2$

- $$\begin{aligned} var^*(X, x) &:= [x] \\ tt^* &:= [in_1(*)] \\ ff^* &:= [in_2(*)] \\ if^*(X, c, c_1, c_2) &:= \text{let } x \Leftarrow c \text{ in } \delta(\text{case } x \text{ of } c_1 | c_2) \\ ab^*(X_1, X_2, f) &:= [f] \\ ap^*(X_1, X_2, c, c_1) &:= \text{let } f, x \Leftarrow c, c_1 \text{ in } \delta(fx) \\ pap^*(X_1, X_2, c, c_1) &:= \text{let } \langle f, x\rangle \Leftarrow pand(c, c_1) \text{ in } \delta(fx) \\ por^* &:= por \end{aligned}$$

– erase δ to recover translation of PL_{fun}

– variations: place δ differently

Monadic approach: caveat

When PL is complex, translation $ML_T(\Sigma) \to ML$ gets complicated.

Incremental approach: basic idea

- adapt techniques and facilities used in ADT
  $ML_T(\Sigma) \to ML$ as implementation of an ADT
- $ML_T(\Sigma) \to ML$ via sequence of steps

$$ML_T(\Sigma_{i+1}) \to ML_T(\Sigma_i) \text{ with } \Sigma_i < \Sigma_{i+1}$$

Need parameterized *polymorphic* translations

$I \in \Pi\Sigma.ML_T(\Sigma_{par} + \Sigma + \Sigma_{new}) \to ML_T(\Sigma_{par} + \Sigma)$

- I polymorphic in signature Σ
- I may have parameter of fixed signature Σ_{par}

I_Σ: implementation of Σ (and parameter) $\mapsto$
reimplementation of Σ and extension to Σ_{new}

* reimplementation of Σ needed, because T changes!

Decomposition of I

- I_{new}: $ML_T(\Sigma_{par} + \Sigma_{new}) \to ML_T(\Sigma_{par})$
  definition of new symbols and redefinition of T
- I_{op}: $ML_T(\Sigma_{op}) \to ML_T(\Sigma_{par} + \Sigma_{op})$
  redefinition of old symbol *op* in *isolation*
  (consistently with redefinition of T)

Reformulation of I in LF

- LF-signature extensions Σ_{par} and Σ_{new}
  i.e. $\Sigma_{ML}, \Sigma_{par}, \Sigma_T, \Sigma_{new} \vdash$ signature
- LF-signature realization

  $I: \Sigma_{ML}+\Sigma_{par}+\Sigma_T+\Sigma_{op}+\Sigma_{new} \to \Sigma_{ML}+\Sigma_{par}+\Sigma_T+\Sigma_{op}$

  where $\Sigma_{ML}, \Sigma_T, \Sigma_{op} \vdash$ signature
  $\Sigma_{ML} + \Sigma_{par}$ (and types A, B in Σ_{op}) unchanged by I

for simplicity

- consider only the Σ_{op}

  $A, B\colon Type$
  old $op\colon \Pi X\colon Type.A, (B \to TX) \to TX$
- ignore axiom part of signatures.

Examples of translation

- I_{se} for adding side-effects
- I_{ex} for adding exceptions
- I_{co} for adding complexity
- I_{con} for adding continuations

Translation I_{se} for adding side-effects

- Σ_{par} parameter symbols

  states $S\colon Type$

- Σ_{new} new symbols

  lookup $lkp\colon TS$

  update $upd\colon S \to T1$

- realization

$$\begin{array}{rl}
T^*X\colon= & S \to T(X\times S) \\
val^*(X,x)\colon= & \lambda s\colon S.[\langle x,s\rangle] \\
let^*(X,Y,f,c)\colon= & \lambda s\colon S.\mathrm{let}\ \langle x,s'\rangle\Leftarrow c(s)\ \mathrm{in}\ f(x,s') \\
\hline
lkp^*\colon= & \lambda s\colon S.[\langle s,s\rangle] \\
upd^*(s)\colon= & \lambda s'\colon S.[\langle *,s\rangle] \\
\hline
op^*(X,a,f)\colon= & \lambda s\colon S.op(X\times S,a,\lambda b\colon B.f(b,s))
\end{array}$$

Remark: must replace an operation $op'\colon A \to TB$ with

$$op(X,a,f) = \mathrm{let}\ b\Leftarrow op'(a)\ \mathrm{in}\ f(b)$$

$op\colon \Pi X\colon Type.A,(B \to TX) \to TX$ fits into Σ_{op}.

Translation I_{ex} for adding exceptions

- Σ_{par} for parameter symbols

  exceptions $E\colon Type$

- Σ_{new} for new symbols

  raise $rse\colon \Pi X\colon Type.E \to TX$

  handle $hdl\colon \Pi X\colon Type.(E \to TX), TX \to TX$

- realization

$$
\begin{array}{rl}
T^*X\colon= & T(X+E) \\
val^*(X,x)\colon= & [in_1(x)] \\
let^*(X,Y,f,c)\colon= & \text{let } u \Leftarrow c \text{ in (case } u \text{ of } x.f(x) | n.rse^*(Y,n)) \\
\hline
rse^*(X,n)\colon= & [in_2(n)] \\
hdl^*(X,f,c)\colon= & \text{let } u \Leftarrow c \text{ in (case } u \text{ of } x.val^*(X,x) | n.f(n)) \\
\hline
op^*(X)\colon= & op(X+E)
\end{array}
$$

  redefinition of op applies to

  $t\colon Type \to Type$

  old $op\colon \Pi X\colon Type.t(TX)$

Remark: improper (but safe) use of new symbol on rhs.

Translation I_{co} for adding complexity

- Σ_{par} for parameter symbols

  monoid $M : Type$

  $1 : M$

  $* : M, M \to M$

  we write $m * n$ for $*(m, n)$

- Σ_{new} for new symbols

  cost $\delta : M \to T1$

- realization

$$
\begin{array}{rl}
T^*X := & T(X \times M) \\
val^*(X, x) := & [\langle x, 1 \rangle] \\
let^*(X, Y, f, c) := & \text{let } \langle x, m \rangle, \langle y, n \rangle \Leftarrow c, f(x) \text{ in } [\langle y, m * n \rangle] \\
\hline
\delta^*(m) := & [\langle *, m \rangle] \\
\hline
op^*(X) := & op(X \times M)
\end{array}
$$

Remark: add to Σ_{par} axioms for monoid, otherwise cannot reprove the axioms in Σ_T.

Translation I_{con} for adding continuations

- Σ_{par} for parameter symbols

  results $R\colon Type$

- Σ_{new} for new symbols

  abort $abort\colon \Pi X\colon Type.R \to TX$

  call-cc $call_{cc}\colon \Pi X, Y\colon Type.((X \to TY) \to TX) \to TX$

- realization

$$
\begin{array}{rl}
T^*X\colon= & (X \to TR) \to TR \\
val^*(X, x)\colon= & \lambda k.k(x) \\
let^*(X, Y, f, c)\colon= & \lambda k.c(\lambda x\colon X.f(x)k) \\
\hline
abort^*(X, r)\colon= & \lambda k.[r] \\
call_{cc}{}^*(X, Y, f)\colon= & \lambda k.f(\lambda x\colon X.\lambda k'.abort^*(X, kx))k \\
\hline
op^*(X, a, f)\colon= & \lambda k.op(R, a, \lambda b\colon B.f(b)k)
\end{array}
$$

Remark: $call_{cc}$ does not fit in Σ_{op}!

I_{ex}: validation of equations in Σ_T

- $let(X, Y, f, val(X, x)) = f(x)$

  let $u \Leftarrow [in_1(x)]$ in (case u of $x.f(x) | n.rse^*(Y, n)$)

  case $in_1(x)$ of $x.f(x) | n.rse^*(Y, n) > f(x)$

- $let(X, X, val(X), c) = c$

  let $u \Leftarrow c$ in (case u of $x.[in_1(x)] | n.[in_2(n)]$)

  let $u \Leftarrow c$ in $[u] > c$

- $let(Y, Z, g, let(X, Y, f, c)) = let(X, Z, let(Y, Z, g) \circ f, c)$

  let $v \Leftarrow$(let $u \Leftarrow c$ in (case u of $x.f(x) | n.[in_2(n)]$)) in
  case v of $y.g(y) | n.[in_2(n)]$

  let $u \Leftarrow c$ in
  let $v \Leftarrow$(case u of $x.f(x) | n.[in_2(n)]$) in
  case v of $y.g(y) | n.[in_2(n)]$

  let $u \Leftarrow c$ in case u of
  x.let $v \Leftarrow f(x)$ in (case v of $y.g(y) | n.[in_2(n)]$)|
  n.let $v \Leftarrow [in_2(n)]$ in (case v of $y.g(y) | n.[in_2(n)]$)

  let $u \Leftarrow c$ in case u of
  x.let $v \Leftarrow f(x)$ in (case v of $y.g(y) | n.[in_2(n)]$)|
  n.(case $in_2(n)$ of $y.g(y) | n.[in_2(n)]$)

  let $u \Leftarrow c$ in case u of
  x.let $v \Leftarrow f(x)$ in (case v of $y.g(y) | n.[in_2(n)]$)|
  $n.[in_2(n)]$

I_{ex}: properties of rse and hdl

- $let(X, Y, f, rse(X, n)) = rse(Y, n)$

  let $u \Leftarrow [in_2(n)]$ in (case u of $x.f(x) | n.rse^*(Y, n)$)

  case $in_2(n)$ of $x.f(x) | n.rse^*(Y, n) > [in_2(n)]$

- $hdl(X, f, val(X, x)) = val(X, x)$

  let $u \Leftarrow [in_1(x)]$ in (case u of $x.val^*(X, x) | n.f(n)$)

  case $in_1(x)$ of $x.val^*(X, x) | n.f(n) > [in_1(x)]$

- $hdl(X, f, rse(X, n)) = f(n)$

  let $u \Leftarrow [in_2(n)]$ in (case u of $x.val^*(X, x) | n.f(n)$)

  case $in_2(n)$ of $x.val^*(X, x) | n.f(n) > f(n)$

- $hdl(X, g, hdl(X, f, c)) = hdl(X, hdl(X, g) \circ f, c)$

  let $v \Leftarrow$(let $u \Leftarrow c$ in (case u of $x.[in_1(x)] | n.f(n)$)) in
  case v of $x.[in_1(x)] | n.g(n)$

  let $u \Leftarrow c$ in
  let $v \Leftarrow$(case u of $x.[in_1(x)] | n.f(n)$) in
  case v of $x.[in_1(x)] | n.g(n)$

  let $u \Leftarrow c$ in case u of
  $x.$let $v \Leftarrow [in_1(x)]$ in (case v of $x.[in_1(x)] | n.g(n)$)|
  $n.$let $v \Leftarrow f(n)$ in (case v of $x.[in_1(x)] | n.g(n)$)

  let $u \Leftarrow c$ in case u of
  $x.$(case $in_1(x)$ of $x.[in_1(x)] | n.g(n)$)|
  $n.$let $v \Leftarrow f(n)$ in (case v of $x.[in_1(x)] | n.g(n)$)

  let $u \Leftarrow c$ in case u of
  $x.[in_1(x)]$|
  $n.$let $v \Leftarrow f(n)$ in (case v of $x.[in_1(x)] | n.g(n)$)

I_{ex}: redefining and reproving

- old operation *op* and old proof *ax*

$$
\begin{array}{ccc}
\begin{array}{l} p\colon \Pi X\colon Type.t(X) \to Prop \\ t\colon Type \to Type \end{array} & \longrightarrow & \begin{array}{l} ax\colon \forall X\colon Type.p(TX)(op\ X) \\ op\colon \Pi X\colon Type.t(TX) \end{array} \\
\uparrow & & \uparrow \\
\Sigma_{ML} & \longrightarrow & \Sigma_T
\end{array}
$$

parameters t and p independent from Σ_T

- redefinition of *op*

  $op^*(X) := op(X+E) \colon t(T(X+E)) = t(T^*X)$

- revalidation of property

  $ax^*(X) := ax(X+E) \colon p(T^*X)(op^*X)$

Remark: easy extension to many *op* and *ax*.

An example

- $t(X) = X, X \to X$, therefore

  $op\colon \Pi X.TX, TX \to TX$

- $p_1(X, f) = \forall x\colon X.f(x, x) = x$

  ax_1X proves that $op\ X$ is idempotent

- $p_2(X, f) = \forall x_1, x_2\colon X.f(x_1, x_2) = f(x_2, x_1)$

  ax_2X proves that $op\ X$ is commutative

- $p_3(X, f) = \forall x_1, x_2, x_3\colon X.f(f(x_1, x_2), x_3) = f(x_1, f(x_2, x_3))$

  ax_3X proves that $op\ X$ is associative

I_{se}: properties of upd and lkp

$upd\colon \Pi X\colon Type.S, TX \to TX$
$lkp\colon \Pi X\colon Type.(S \to TX) \to TX$

- $upd(s, lkp(f)) = upd(s, fs)$
- $upd(s_1, upd(s_2, c)) = upd(s_2, c)$
- $lkp(\lambda s.upd(s, fs)) = lkp(f)$
- $lkp(\lambda s.c) = c$

I_{se}: redefining and reproving

$op\colon \Pi X\colon Type.t(TX)$ where $tX = (A, (B \to X) \to X)$

- redefinition of op

  $op^*(X, a, f) = \lambda s.op(X{\times}S, a, \lambda b.fbs)\colon t(T^*X)$
- $\forall X.p(TX)(op\ X)$ where

  $p\colon \Pi X\colon Type.t(X) \to Prop$ s.t.

  $\forall X, S\colon Type.\forall op\colon t(X).$
  $p(X)(op) \supset p(S \to X)(\lambda a, f.\lambda s.op(a, \lambda b.fbs))$
- op **distributes** over let, i.e.

  $let(f, op(a, h)) = op(a, \lambda b.let(f, hb))$

I_{se}: interaction between old and new ops

op distributes over upd and lkd

- $upd(s, op(a, h)) = op(a, \lambda b.upd(s, hb))$
- $lkp(\lambda s.op(a, \lambda b.hbs)) = op(a, \lambda b.lkp(\lambda s.hbs))$

I_{se}: preservation of alg. equations

property $\forall X.p(TX)(op\ \ X)$ of $op\colon \Pi X.t(TX)$ preserved

- provided $t(X) = (X^n \to X)$ and
- $p(X)(f) = \forall \overline{z}.eq(e_1, e_2)$
  where $e \in Exp\colon := \ z \mid f(\overline{e})$

prove $\forall X, S\colon Type.\forall op\colon t(X).$
$$p(X)(op) \supset p(S \to X)(\lambda\overline{x}.\lambda s.op(\overline{x_i s}))$$

Hint

prove (by induction on e) that

$$(e[\overline{z}, f\colon = \overline{x}, \lambda\overline{x}.\lambda s.op(\overline{x_i s})])s = e[\overline{z}, f\colon = \overline{x_i s}, op]$$

where $op\colon t(X)$, $s\colon S$ and $x_i\colon S \to X$

- case e is z_i
  $lhs = x_i s = rhs$
- case e is $f(\overline{e})$
  $lhs =$
  $(\lambda\overline{x}.\lambda s.op(\overline{x_i s})(\overline{e}[\overline{z}, f\colon = \ldots])s =$
  $op(\overline{e_i[\overline{z}, f\colon = \ldots]s}) =$ by IH
  $op(\overline{e}[\overline{z}, f\colon = \ldots]) =$
  rhs

Incremental approach at work

Basic translations

- $I_{se}TX = S \to T(X \times S)$ adding side-effects
- $I_{ex}TX = T(X + E)$ adding exceptions
- $I_{co}TX = T(X \times M)$ adding complexity
- $I_{con}TX = (X \to TR) \to TR$ adding continuations

* $I_{res}TX = \mu X'.T(X + X')$ adding resumptions

Composite translations

- $I_{ex}(I_{se}T)X = S \to T((X + E) \times S)$

  imperative language with exceptions, e.g. SML

- $I_{se}(I_{ex}T)X = S \to T((X \times S) + E)$

  imperative languages with recovery blocks: errors handled by executing alternative code from a check-point

- $I_{se}(I_{con}T)X = (X \to S \to TR) \to S \to TR$

  imperative languages with goto

  $I_{con}(I_{se}T)X$, as above but R replaced by $R \times S$

- $I_{res}(I_{se}T)X = \mu X'.S \to T((X + X') \times S)$

  parallel imperative languages

- $I_{se}(I_{res}T)X = S \to \mu X'.T((X \times S) + X')$

  transaction based languages, state changes happen only after interaction completed successfully.

Incremental approach at work
an example

Basic translations

- $I_{se}TX = S \to T(X \times S)$ adding side-effects
- $I_{ex}TX = T(X + E)$ adding exceptions

Composite translations

- $I_{ex}(I_{se}T)X = S \to T((X + E) \times S)$
  imperative languages with exceptions like ML
  - properties of *rse* and *hdl*
  - properties of *lkp* and *upd* (preserved)
  - distributivity of *rse* w.r.t. *let*
  - distributivity of *lkp* and *upd* w.r.t. *let* (preserved)
  - distributivity of *lkp* and *upd* w.r.t. *hdl*
- $I_{se}(I_{ex}T)X = S \to T((X \times S) + E)$
  imperative languages with recovery blocks: errors handled by executing alternative code from a checkpoint
  - properties of *rse* and *hdl* (preserved)
  - properties of *lkp* and *upd*
  - distributivity of *rse* w.r.t. *let* (preserved)
  - distributivity of *lkp* and *upd* w.r.t. *let*
  - distributivity of *rse* and *hdl* w.r.t. *lkp* and *upd*

Properties

- properties of lkp and upd

  $upd(s, lkp(f)) = upd(s, fs)$

  $upd(s_1, upd(s_2, c)) = upd(s_2, c)$

  $lkp(\lambda s.upd(s, fs)) = lkp(f)$

  $lkp(\lambda s.c) = c$

- properties of rse and hdl

  $hdl(f, val(x)) = val(x)$

  $hdl(f, rse(n)) = f(n)$

  $hdl(rse, c) = c$

  $hdl(g, hdl(f, c)) = hdl(hdl(g) \circ f, c)$

Distributivity

- op **distributes** over $let \overset{\Delta}{\Longleftrightarrow}$

  $let(f, op(a, h)) = op(a, \lambda b.let(f, hb))$

- distributivity of rse w.r.t. let

  $let(f, rse(n)) = rse(n)$

- distributivity of lkp and upd w.r.t. let

Properties of I_{se}

- *op* distributes w.r.t. *upd* and *lkd*

  $upd(s, op(a, h)) = op(a, \lambda b.upd(s, hb))$

  $lkp(\lambda s.op(a, \lambda b.hbs)) = op(a, \lambda b.lkp(\lambda s.hbs))$

- preserves *algebraic* properties
- preserves distributivity w.r.t. *let*

Properties of I_{ex}

- maps *op* distributing w.r.t. *let*

  to *op* distributing w.r.t. *hdl*

  $hdl(f, op(a, h)) = op(a, \lambda b.hdl(f, hb))$

- preserves *logical* properties
- preserves distributivity w.r.t. *let*

Operations for $I_{ex}(I_{se}T)$

$$
\begin{array}{rl}
T^*X{:=} & S \to T((X+E)\times S) \\
\hline
lkp^*(f){:=} & \lambda s{:}\,S.fss \\
upd^*(s,c){:=} & \lambda s'{:}\,S.cs \\
\hline
rse^*(n){:=} & \lambda s.[\langle in_2(n), s\rangle] \\
hdl^*(f,c){:=} & \lambda s.\text{let } \langle u, s'\rangle \Leftarrow cs \text{ in} \\
 & \quad\quad \text{case } u \text{ of } x.[\langle in_1(x), s'\rangle] | n.f(n)s'
\end{array}
$$

- $hdl(f, upd(s,c)) = upd(s, hdl(f,c))$
- $hdl(f, lkp(\lambda s.hs)) = lkp(\lambda s.hdl(f, hs))$

Operations for $I_{se}(I_{ex}T)$

$$
\begin{array}{rl}
T^*X{:=} & S \to T((X\times S)+E) \\
\hline
lkp^*(f){:=} & \lambda s{:}\,S.fss \\
upd^*(s,c){:=} & \lambda s'{:}\,S.cs \\
\hline
rse^*(n){:=} & \lambda s.[in_2(n)] \\
hdl^*(f,c){:=} & \lambda s.\text{let } u \Leftarrow cs \text{ in} \\
 & \quad\quad \text{case } u \text{ of } \langle x, s'\rangle.[in_1(\langle x, s'\rangle)] | n.f(n)s
\end{array}
$$

- $upd(s, rse(n)) = rse(n)$
- $lkp(\lambda s.rse(n)) = rse(n)$
- $upd(s, hdl(f,c)) = hdl(\lambda n.upd(s, fn), upd(s,c))$
- $lkp(\lambda s.hdl(\lambda n.fsn, cs)) = hdl(\lambda n.lkp(\lambda s.fsn), lkp(\lambda s.cs))$

Some Remarks

- monadic approach of limited use, when computation takes place only at ground types
  e.g. in PCF-like languages (Algol)
- satisfactory treatment of PL_{par} requires
  - fix-point operator
  - inductive types
  - translation for adding resumptions
- encoding in LF depends on what one wants to do:
  - substitution
  - induction of syntax
- axiomatization of types for ML:
  - categorical properties of universal constructions
  - intro-elim rules for inductive types in TT

References

[CM93] P. Cenciarelli and E. Moggi. A syntactic approach to modularity in denotational semantics. In *Proceedings of the Conference on Category Theory and Computer Science, Amsterdam, Sept. 1993*, 1993. CWI Tech. Report.

[CP88] T. Coquand and C. Paulin. Inductively defined types. In *Conference on Computer Logic*, volume 417 of *LNCS*. Springer Verlag, 1988.

[CP92] R.L. Crole and A.M. Pitts. New foundations for fixpoint computations: Fix hyperdoctrines and the fix logic. *Information and Computation*, 98, 1992.

[Fre90] P. Freyd. Recursive types reduced to inductive types. In J. Mitchell, editor, *Proc. 5th Symposium in Logic in Computer Science*, Philadelphia, 1990. I.E.E.E. Computer Society.

[Fre92] P. Freyd. Algebraically complete categories. In A. Carboni, M.C. Pedicchio, and G. Rosolini, editors, *Category Theory '90*, volume 1144 of *Lecture Notes in Mathematics*, Como, 1992. Springer-Verlag.

[Geu92] H. Geuvers. The church-rosser property for $\beta\eta$-reduction in typed lambda calculi. In A. Scedrov, editor, *Proc. 7th Symposium in Logic in Computer Science*, Santa Cruz, 1992. I.E.E.E. Computer Society.

[Gor79] M.J.C. Gordon. *The Denotational Description of Programming Languages*. Springer-Verlag, 1979.

[GS89] C. Gunter and D.S. Scott. Semantic domains. Technical Report MS-CIS-89-16, Dept. of Comp. and Inf. Science, Univ. of Pennsylvania, 1989. to appear in North Holland Handbook of Theoretical Computer Science.

[GW94] H. Geuvers and B. Werner. On the church-rosser property for expressive type systems and its consequences for their metatheory. In S. Abramsky, editor, *Proc. 9th Symposium in Logic in Computer Science*, Paris, 1994. I.E.E.E. Computer Society.

[HHP87] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In R. Constable, editor, *Proc. 2th Symposium in Logic in Computer Science*, Ithaca, NY, 1987. I.E.E.E. Computer Society.

[Luo94] Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science*. International Series of Monographs on Computer Science. Oxford University Press, 1994.

[MC88] A.R. Meyer and S.S. Cosmodakis. Semantic paradigms: Notes for an invited lecture. In *3rd LICS Conf.* IEEE, 1988.

[Mog91] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1), 1991.

[Mos89] P. Mosses. Denotational semantics. Technical Report DAIMI-PB-276, CS Dept., Aarhus University, 1989. to appear in North Holland Handbook of Theoretical Computer Science.

[Mos90a] P. Mosses. *Action Semantics*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[Mos90b] P. Mosses. Denotational semantics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. North Holland, 1990.

[Sch86] D.A. Schmidt. *Denotational Semantics: a Methodology for Language Development*. Allyn & Bacon, 1986.

[Sco93] D.S. Scott. A type-theoretic alternative to CUCH, ISWIM, OWHY. *Theoretical Computer Science*, 121, 1993.

[Sim92] A.K. Simpson. Recursive types in kleisli categories. available via FTP from theory.doc.ic.ac.uk, 1992.

[SP82] M. Smyth and G. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal of Computing*, 11, 1982.

[Ten91] R.D. Tennent. *Semantics of Programming Languages*. Prentice Hall, 1991.