Typed Intermediate Languages in the IML Compiler

The IML compiler team

November 19, 2004

Contents

1 Introduction

1.1 Notation

First of all, the following are some of the notations used frequently.

- i, j, k, n, and m are integers.
- con, $label$, and id are strings to denote the name of a (type) constructor, a label, an identifier.
- $\{\ label_1 = object_1,\ \ldots\ , label_n = object_n\ \}$ denotes a record.
- $[\ object_1,\ \ldots\ , object_n\ ]$ denotes a list of objects.
- $\{\ label_1 \mapsto object_1,\ \ldots\ , label_n \mapsto object_n\ \}$ denotes a mapping of labels into objects.

Second, symbols for syntactic and semantic objects are chosen to have the same name as (or, the abbreviated name of) the actual name of types and constructors for the objects' representation in the implementation (i.e., the IML compiler). For example,

- ty is chosen to denote a type, and $FUNty\ (ty_1, ty_2)$ is chosen to denote a function type. The names are derived from the following ML datatype declaration for the types' representation in the actual implementation.

  – datatype ty = $\cdots$ | FUNty of ty * ty | $\cdots$

- *expopt* denotes an optional expression, which is represented by the ML type " exp option " where exp is the ML type for term representation.
- *tylist* denotes a list of types (" ty list "), and $fieldtys$ denotes a mapping of field labels to types (" ty SEnv.map ")[1]. Recall that labels are represented by strings. *patexplist* denotes a list of pairs of a pattern and an expression (" (pat *exp) list ").

Third, the various kinds of rules we use have the form as

[1] Although some extra definitions of functors and structures in the ML library are needed to explain what "ty SEnv.map" is, this can be easily read as an ML type for mappings of strings to types.

$$\text{(rule)}\ \frac{Judgment_1 \quad \cdots \quad Judgment_n}{Judgment}$$

followed by some extra conditions, whenever they are needed to be stated, after "where" clause.

2 Typed Pattern Language

A specification for the typed pattern language (TP), which is an intermediate language of the IML compiler, is defined.

2.1 Types and terms

For the syntax of types and kinds, we refer to the datatypes ty, eqkind, reckind, and rank in "types/main/types.ppg." For the syntax of terms, we refer to the datatypes tpexp, tpdecl, and tppat in "typedcalc/main/typedcalc.ppg."

Each new name for types and terms will be explained whenever it appears for the first time, or we will skip explaining it when it is too obvious.

2.2 Typing environments

A typing environment *env* consists of four sub-environments: a type constructor environment *tcenv*, a global variable environment *genv*, a bound type variable environment *btvenv*, and a variable environment *varenv*.

$$\begin{aligned}
env &::= \{tcenv = tcenv,\ genv = varenv,\ btvenv = btvenv,\ varenv = varenv\} \\
tcenv &::= \{con_1 \mapsto tybindinfo_1,\ \ldots, con_n \mapsto tybindinfo_n\} \\
genv &::= \{id_1 \mapsto idstate_1,\ \ldots, id_n \mapsto idstate_n\} \\
btvenv &::= \{1 \mapsto btvkind_1,\ \ldots, n \mapsto btvkind_n\} \\
varenv &::= \{id_1 \mapsto idstate_1,\ \ldots, id_n \mapsto idstate_n\}
\end{aligned}$$

A type constructor environment ($tcenv$) is a mapping of type constructor names (con) into type binding information descriptions ($tybindinfo$), which are either a datatype declaration or a type synonym declaration. A datatype declaration is described by a name, an arity, an identification number, an equality kind, a data constructor environment.

$$\begin{aligned} tybindinfo &::= TYCON\ tycon \mid TYFUN\ \{name : string, tyargs : btvenv, body : ty\} \\ tycon &::= \{name : string, arity : int, id : int, eqkind : eqkind\_ref, datacon : varenv\_ref\} \end{aligned}$$

A variable environment ($genv$ or $varenv$) is a mapping of identifies into identifier states, which are one of a variable, a data constructor, a primitive operator, and an overloaded primitive operator. Basically, each identifier is described by its name and type. $opriminfo$ not only contains a name and a type for an overloaded primitive operator, but it also contains another names and types for each instance primitive operator obtained from resolving the overloading. $coninfo$ describes a data constructor by its name, type, the availability of a constructor argument ($funtycon$), an extra tag when it is an exception constructor, and a datatype declaration ($tycon$) where it belongs.

$$\begin{aligned} idstate &::= VARID\ varinfo \mid CONID\ coninfo \mid PRIM\ priminfo \mid OPRIM\ opirminfo \\ varinfo &= \{name : string,\ ty : ty\} \\ priminfo &= \{name : string,\ ty : ty\} \\ opriminfo &= \{name : string,\ ty : ty,\ instances : priminfos\} \\ coninfo &= \{name : string,\ ty : ty,\ funtycon : bool,\ exntag : int,\ tycon : tycon\} \end{aligned}$$

A bound type variable environment ($btvenv$) is a mapping of de Bruijn indexes (i.e., integers) into bound type kinds, Each $btvkind$ includes a record kind, an equality kind, and a rank.

$$\begin{aligned} btvkind &= \{\ reckind : reckind,\ eqkind : eqkind,\ rank : bool,\ index : int\ \} \\ eqkind &= EQ \mid NONEQ \end{aligned}$$

$$reckind \quad = \quad UNIV \mid REC\ \{label_1 \mapsto ty_1, ..., label_n \mapsto ty_n\} \mid OVERLOADED\ tylist$$

Selection and extension operations on typing environments are defined as follows:

- $\#varenv(env)$ selects the variable environment of the typing environment.
- $env + varenv'$ extends env by extending the current variable environment $\#varenv(env)$ with an additional variable environment $varenv'$.
- The selection and extension operations for $tcenv$, $genv$, and $btvenv$ are defined similarly.

2.3 Typing rules

2.3.1 Well-formed types

$\boxed{env \rhd ty}$

- A type ty is closed under a bound-type-variable environment $\#btvenv(env)$.

$\boxed{env \rhd tylist}$

- Each type of $tylist$ is closed under a bound-type-variable environment $\#btvenv(env)$.

2.3.2 Type equality

$\boxed{env \rhd ty \ = \ ty'}$

- A type ty is equal to another type ty' under a bound-type-variable environment $\#btvenv(env)$.
- This judgment is assumed to be implicitly used for each pair of multiple occurrences of any type in kinding rules and typing rules.

2.3.3 Kinding types

$\boxed{env \rhd ty : eqk}$

- A type ty has equality-kind eqk under a bound-type-variable environment $\#btvenv(env)$.

(eq-errorty) $$\frac{}{env \rhd ERRORty : NONEQ}$$

(eq-dummyty) $$\frac{}{env \rhd DUMMYty : NONEQ}$$

(eq-tyvarty-tvar) $$\frac{tvkind \;=\; \{reckind = \_\,,\; eqkind = eqk,\; tyvarname = \_\,, id = \_\,\}}{env \rhd TYVARty\ (ref\ (TVAR\ tvkind)) : eqk}$$

(eq-tyvar-subst) $$\frac{env \rhd ty : eqk}{env \rhd TYVARty\ (ref\ (SUBSTITUTED\ ty)) : eqk}$$

(eq-boundvarty) $$\frac{\#btvenv(env)\ (i) \;=\; \{reckind = \_\,,\; eqkind = eqk, rank = \_\,, index = \_\,\}}{env \rhd BOUNDVARty\ i : EQ}$$

- Every bound type variable is assumed to have eqkind EQ.

(eq-funty) $$\frac{}{env \rhd FUNty\ (ty_1, ty_2) : NONEQ}$$

(eq-iabsty) $$\frac{}{env \rhd IABSty\ (tylist, ty) : NONEQ}$$

(eq-recordty) $$\frac{env \rhd ty_i : eqk_i \quad (1 \;\leq i \leq\; n)}{env \rhd RECORDty\ \{label_1 \mapsto ty_1, ..., label_n \mapsto ty_n\} : eqk}$$

- where
  - $eqk \;=\; EQ$ if $eqk_i = EQ$ for all i. Otherwise, $eqk \;=\; NONEQ$.

(eq-conty) $$\frac{env \rhd ty_i : eqk_i \quad (1 \leq i \leq \text{the length of } tylist)}{env \rhd CONty\ \{tycon = tycon,\; args = tylist\} : eqk}$$

- where
  - $tycon \;=\; \{name = \_,\; arity = \_,\; id = \_,\; eqkind = eqk\_ref,\; datacon = \_\,\}$
  - $eqk \;=\; eqk\_ref$ if $eqk_i = EQ$ for all i. Otherwise, $eqk \;=\; NONEQ$.

(eq-polyty) $$\frac{}{env \rhd POLYty\ \{boundtvars = btvenv,\; body = ty\} : NONEQ}$$

(eq-boxedty) $$\frac{}{env \rhd BOXEDty : NONEQ}$$

(eq-atomty) $$\frac{}{env \rhd ATOMty : NONEQ}$$

(eq-indexty) $$\frac{env \rhd ty : REC\ fieldtys \quad fieldtys(l)\ =\ ty' \quad env \rhd ty' : eqk}{env \rhd INDEXty\ (ty, l) : eqk}$$

(eq-bmsabsty) $$\frac{}{env \rhd BMSABSty\ (tylist, ty) : NONEQ}$$

(eq-bitmapty) $$\frac{}{env \rhd BITMAPty\ [\ bitty_1,\ \ldots\ ,\ bitty_n\ ] : NONEQ}$$

$\boxed{env \rhd ty : reck}$

- A type ty has record-kind $reck$ under a bound-type-variable environment $\#btvenv(env)$.

(rec-errorty) $$\frac{}{env \rhd ERRORty : UNIV}$$

(rec-dummyty) $$\frac{}{env \rhd DUMMYty : UNIV}$$

(rec-tyvarty-tvar) $$\frac{tvkind\ =\ \{reckind = reck,\ eqkind = \_\ ,\ tyvarname = \_\ ,\ id = \_\ \}}{env \rhd TYVARty\ (ref\ (TVAR\ tvkind)) : reck}$$

(rec-tyvar-subst) $$\frac{env \rhd ty : reck}{env \rhd TYVARty\ (ref\ (SUBSTITUTED\ ty)) : reck}$$

(rec-boundvarty) $$\frac{\#btvenv(env)\ (i)\ =\ \{reckind = reck,\ eqkind = \_,\ rank = \_,\ index = \_\ \}}{env \rhd BOUNDVARty\ i : reck}$$

(rec-funty) $$\frac{}{env \rhd FUNty\ (ty, ty') : UNIV}$$

(rec-iabsty) $$\frac{}{env \rhd IABSty\ (tylist, ty) : UNIV}$$

(rec-recordty) $$\frac{}{env \rhd RECORDty\ \{label_1 \mapsto ty_1, ..., label_n \mapsto ty_n\} : REC\ \{label_1 \mapsto ty_1, ..., label_n \mapsto ty_n\}}$$

(rec-conty) $$\frac{}{env \rhd CONty\ \{tycon = tycon,\ args = tylist\} : UNIV}$$

(rec-polyty) $$\frac{}{env \rhd POLYty\ \{boundtvars = btvenv,\ body = ty\} : UNIV}$$

(rec-boxedty) $$\frac{}{env \rhd BOXEDty : UNIV}$$

(rec-atomty) $$\frac{}{env \rhd ATOMty : UNIV}$$

(rec-indexty) $$\frac{env \rhd ty : REC\ fieldtys \quad fieldtys(l)\ =\ ty' \quad env \rhd ty' : reck'}{env \rhd INDEXty\ (ty, l) : reck'}$$

(rec-bmsabsty) $$\frac{}{env \rhd BMSABSty\ (tylist, ty) : UNIV}$$

(rec-bitmapty) $$\frac{}{env \rhd BITMAPty\ [\ bitty_1,\ ...\ ,\ bitty_n\ ] : UNIV}$$

2.3.4 Type instantiations

$\boxed{eqk\ \leq\ eqk'}$

- $eqk \leq eqk'$ permits the substitution of a bound type variable of eqkind eqk with a type of eqkind eqk'.

  (ee) $EQ\ \leq\ EQ$

  (ne) $NONEQ\ \leq\ EQ$

  (nn) $NONEQ\ \leq\ NONEQ$

$\boxed{reck \ \leq \ reck'}$

- $reck \leq reck'$ permits the substitution of a bound type variable of reckind $reck$ with a type of reckind $reck'$.

(uu) $UNIV \ \leq \ UNIV$

(ur) $UNIV \ \leq \ REC \ fieldtys$

(rr) $REC \ \{label_1 \mapsto ty_1, ..., label_n \mapsto ty_n\} \ \leq \ REC \ \{label_1 \mapsto ty_1, ..., label_n \mapsto ty_n, \ ... \ \}$

$\boxed{env \rhd instantiate \ (ty, \ tylist) \ \Rightarrow \ ty'}$

(inst-poly)
$$\frac{env \rhd POLYty \ (btvenv, ty) \quad env \rhd tylist \qquad env \rhd ty_i : eqk'_i \quad env \rhd ty_i : reck'_i \quad (1 \leq i \leq n)}{env \rhd instantiate \ (POLYty \ (btvenv, ty), \ tylist) \ \Rightarrow \ ty[ty_1/btv_1, ..., ty_n/btv_n]}$$

- where
  - $tylist \ = \ [ty_1, \ ... \ , ty_n]$
  - $btvenv \ = \ \{1 \mapsto btv_1, \ ... \ , n \mapsto btv_n\}$
  - $btv_i \ = \ \{reckind = reck_i, \ eqkind = eqk_i, \ rank = \_, \ index = \_ \ \}$
  - $eqk_i \leq eqk'_i$ and $reck_i \leq reck'_i$ for all i

(inst-mono)
$$\frac{env \rhd ty \quad ty \text{ is not } POLYty \ (\_, \ \_)}{env \rhd instantiate \ (ty, \ []) \ \Rightarrow \ ty}$$

$\boxed{env \rhd overload \ (ty, \ tylist) \ \Rightarrow \ ty'}$

(overload-poly)
$$\frac{env \rhd POLYty \ (btvenv, ty) \quad env \rhd tylist \qquad env \rhd ty_i : eqk'_i \quad ty_i \ \in \ tylist_i \quad (1 \leq i \leq n)}{env \rhd overload \ (POLYty \ (btvenv, ty), \ tylist) \ \Rightarrow \ ty[ty_1/btv_1, ..., ty_n/btv_n]}$$

- where
  - $tylist \ = \ [\ ty_1, \ ... \ , ty_n \ ]$
  - $btvenv \ = \ \{1 \mapsto btv_1, \ ... \ , n \mapsto btv_n\}$
  - $btv_i \ = \ \{reckind = OVERLOADED \ tylist_i, \ eqkind = eqk_i, \ rank = \_, \ index = \_ \ \}$

– $eqk_i \leq eqk'_i$ for all i

$$\text{(overload-mono)}\ \frac{env \rhd ty \quad ty \text{ is not } POLYty\ (\_,\ \_)}{env \rhd instantiate\ (ty,\ [])\ \Rightarrow\ ty}$$

2.3.5 Typing constants

$\boxed{env \rhd constant : ty}$

$$\text{(int)}\ \frac{}{env \rhd INTCONST\ i : intty}$$

- where
  - $intty\ =\ CONty\ \{\ tycon = intTycon,\ args = []\ \}$
  - $intTycon\ =\ \{\ name =$ "int", $arity = 0,\ id = \_,\ eqkind = EQ\,ref,\ datacon = \{\}\,ref\ \}$
  - $\{\}$ is an empty mapping.
- *stringty*, *realty*, *charty*, and *wordty* are similarly defined.

$$\text{(string)}\ \frac{}{env \rhd STRING\ s : stringty}$$

$$\text{(real)}\ \frac{}{env \rhd REAL\ f : realty}$$

$$\text{(char)}\ \frac{}{env \rhd CHAR\ c : charty}$$

$$\text{(word)}\ \frac{}{env \rhd WORD\ w : wordty}$$

2.3.6 Typing expressions

$\boxed{env \rhd exp : ty}$

$$\text{(error)}\ \frac{}{env \rhd ERROR : ERRORty}$$

(constant) $$\frac{env \rhd c : ty}{env \rhd CONSTANT\ c : ty}$$

(var) $$\frac{\#varenv(env)\ (id)\ =\ VARID\ \{name = id,\ ty = ty\}}{env \rhd VAR\ (path,\ \{name = id,\ ty = ty\}) : ty}$$

- $path\ ::=\ Pid\ string\ \mid\ Pdot\ (path,\ path)$

(global) $$\frac{\#genv(env)\ (id)\ =\ VARID\ \{name = id,\ ty = ty\}}{env \rhd GLOBAL\ (path,\ \{name = id,\ ty = ty\}) : ty}$$

(primapply) $$\frac{env \rhd ty \quad env \rhd tylist \quad env \rhd instantiate\ (ty,\ tylist)\ \Rightarrow\ ty_1 \quad env \rhd expopt : ty_1 \Rightarrow ty_2}{env \rhd PRIMAPPLY\ (\{name = id,\ ty = ty\},\ tylist,\ expopt) : ty_2}$$

(oprimapply) $$\frac{env \rhd ty \quad env \rhd tylist \quad env \rhd overload\ (ty,\ tylist)\ \Rightarrow\ ty_1 \quad env \rhd expopt : ty_1 \Rightarrow ty_2}{env \rhd OPRIMAPPLY\ (\{name = id,\ ty = ty\},\ tylist,\ expopt) : ty_2}$$

(construct) $$\frac{env \rhd ty \quad env \rhd tylist \quad env \rhd instantiate\ (ty,\ tylist)\ \Rightarrow\ ty_1 \quad env \rhd expopt : ty_1 \Rightarrow ty_2}{env \rhd CONSTRUCT\ (path,\ coninfo,\ tylist,\ expopt) : ty_2}$$

- where
  - $coninfo\ =\ \{name = id,\ funtycon = funtycon,\ ty = ty,\ exntag = exntag,\ tycon = tycon\ \}$
  - $funtycon = true$ if $expopt = SOME\ exp$, and $funtycon = false$ if $expopt = NONE$.
  - $tycon\ =\ \{name = tyname,\ arity = arity,\ id = id,\ eqkind = eqkind\_ref,\ datacon = datacon\_ref\ \}$
  - $datacon\_ref\ (id)\ =\ CONID\ coninfo$
  - $\#tcenv(env)\ (tyname)\ =\ TYCON\ tycon$

(app) $$\frac{env \rhd FUNty\ (ty_1, ty_2) \quad env \rhd exp_1 : FUNty\ (ty_1, ty_2) \quad env \rhd exp_2 : ty_1}{env \rhd APP\ (exp_1,\ FUNty\ (ty_1, ty_2),\ exp_2) : ty_1}$$

$$\text{(monolet)}\ \frac{env \rhd valbindlist : varenv \quad env + varenv \rhd exp : ty}{env \rhd MONOLET\ (valbindlist, exp) : ty}$$

$$\text{(let)}\ \frac{env \rhd tylist \quad env \rhd decl : (tcenv,\ varenv) \quad env + tcenv + varenv \rhd explist : tylist}{env \rhd LET\ (decl, explist, tylist) : last\ tylist}$$

- where
  - $last\ tylist$ is the last element of $tylist$.

$$\text{(record)}\ \frac{env \rhd ty \quad env \rhd fieldlist : fieldtys}{env \rhd RECORD\ (fieldlist,\ ty) : ty}$$

- where
  - $ty\ =\ RECORDty\ fieldtys$
  - $fieldlist\ =\ \{label_1,\ \cdots,\ label_n\}$
  - $fieldtys\ =\ \{label_1 \mapsto ty_1 \cdots label_n \mapsto ty_n\}$

$$\text{(select)}\ \frac{env \rhd ty : REC\ \{label \mapsto ty'\} \quad env \rhd exp : ty}{env \rhd SELECT\ (label,\ exp,\ ty) : ty'}$$

$$\text{(raise)}\ \frac{env \rhd ty \quad env \rhd exp : exnty}{env \rhd RAISE\ (exp, ty) : ty}$$

$$\text{(handle)}\ \frac{env \rhd exp : ty \quad env + VARID\ \{name = id,\ ty = exnty\} \rhd exp' : ty}{env \rhd HANDLE\ (exp, \{name = id,\ ty = exnty\}, exp') : ty}$$

$$\text{(case)}\ \frac{env \rhd ty \quad env \rhd ty' \quad env \rhd exp : ty \quad env \rhd patexplist : ty \Rightarrow ty'}{env \rhd CASE\ (exp,\ ty,\ patexplist,\ ty',\ casekind) : ty'}$$

- where
  - $casekind\ ::=\ BIND \mid MATCH \mid HANDLE$

$$\frac{env \rhd ty \quad env \rhd ty'}{env + \{id \mapsto VARID\ \{name = id,\ ty = ty\}\} \rhd exp : ty'}\ \text{(fn)}$$
$$env \rhd FN\ (\{name = id,\ ty = ty\},\ ty',\ exp) : ty'$$

$$\frac{env + btvenv \rhd FUNty\ (ty,\ ty') \quad env + btvenv + \{id \mapsto VARID\ \{name = id, ty = ty\}\} \rhd exp : ty'}{env \rhd POLYFN\ (btvenv,\ \{name = id,\ ty = ty\},\ ty',\ exp) : ty_{poly}}\ \text{(polyfn)}$$

- where
  - $ty_{poly}\ =\ POLYty\ \{boundtvars = btvenv, body = FUNty\ (ty, ty')\}$

$$\frac{env + btvenv \rhd ty \quad env + btvenv \rhd exp : ty}{env \rhd POLY\ (btvenv,\ ty,\ exp) : POLYty\ \{boundtvars = btvenv,\ body = ty'\}}\ \text{(poly)}$$

$$\frac{env \rhd ty \quad env \rhd tylist \quad env \rhd exp : ty \quad env \rhd instantiate\ (ty,\ tylist)\ \Rightarrow\ ty'}{env \rhd TAPP\ (exp,\ ty,\ tylist) : ty'}\ \text{(tapp)}$$

$$\frac{env \rhd explist : tylist}{env \rhd SEQ\ (\ explist,\ tylist\ ) : last\ tylist}\ \text{(seq)}$$

- where
  - *last tylist* is the last element of *tylist*.

$\boxed{env \rhd explist : tylist}$

$$\frac{env \rhd exp_i : ty_i \quad (i \leq i \leq n)}{env \rhd [exp_1,\ \ldots\ ,\ exp_n] : [ty_1,\ \ldots\ ,\ ty_n]}\ \text{(explist)}$$

$\boxed{env \rhd expopt : \tau \Rightarrow \tau'}$

$$\frac{}{env \rhd NONE : ty \Rightarrow ty}\ \text{(none-exp)}$$

$$\text{(some-exp)}\ \frac{env \rhd exp : ty}{env \rhd SOME\ exp : ty \Rightarrow ty'}$$

2.3.7 Typing patterns

$\boxed{env \rhd pat : ty,\ varenv}$

$$\text{(patwild)}\ \frac{env \rhd ty}{env \rhd PATWILD\ ty : ty,\ \{\}}$$

$$\text{(patvar)}\ \frac{env \rhd ty}{env \rhd PATVAR\ (varinfo,\ loc) : ty,\ \{id \mapsto VARID\ varinfo\}}$$

- where
  - $varinfo\ =\ \{name = id,\ ty = ty\}$

$$\text{(patconstant)}\ \frac{env \rhd c : ty}{env \rhd PATCONSTANT\ (c,\ ty) : ty,\ \{\}}$$

$$\text{(patconstruct)}\ \frac{\begin{array}{c} env \rhd ty \quad env \rhd ty_{poly} \quad env \rhd tylist \\ env \rhd instantiate\ (ty_{poly},\ tylist)\ \Rightarrow\ ty \\ env \rhd patopt : ty \Rightarrow ty',\ varenv \end{array}}{env \rhd PATCONSTRUCT\ (path,\ coninfo,\ tylist,\ patopt,\ ty) : ty',\ varenv}$$

- where
  - $coninfo\ =\ \{name = id,\ funtycon = funtycon,\ ty = ty_{poly},\ exntag = exntag,\ tycon = tycon\ \}$
  - $funtycon = true$ if $expopt = SOME\ exp$, and $funtycon = false$ if $expopt = NONE$.
  - $tycon\ =\ \{name = tyname,\ arity = arity,\ id = id,\ eqkind = eqkind\_ref,\ datacon = datacon\_ref\ \}$
  - $datacon\_ref\ (id)\ =\ CONID\ coninfo$
  - $\#tcenv(env)\ (tyname)\ =\ TYCON\ tycon$

$$\text{(patrecord)}\quad \frac{env \rhd ty \quad env \rhd ty : REC\ fieldenv \qquad env \rhd patfields : fieldenv,\ varenv}{env \rhd PATRECORD\ (patfields,\ ty) : ty,\ varenv}$$

$$\text{(patlayered)}\quad \frac{env \rhd pat_1 : ty,\ varenv_1 \qquad env \rhd pat_2 : ty,\ varenv_2}{env \rhd PATLAYERED\ (pat_1, pat_2) : ty,\ varenv_1 + varenv_2}$$

- where
  - $pat_1 \ = \ PATVAR\ (varinfo,\ loc)$
  - $dom(varenv_1) \ \cap \ dom(varenv_2) \ = \ \emptyset$

$\boxed{env \rhd patopt : ty \Rightarrow ty',\ varenv}$

$$\text{(pat-none)}\quad \frac{}{env \rhd NONE : ty \Rightarrow ty,\ \{\}}$$

$$\text{(pat-some)}\quad \frac{env \rhd pat : ty,\ varenv}{env \rhd SOME\ pat : FUNty\ (ty, ty') \Rightarrow ty',\ varenv}$$

$\boxed{env \rhd patfields : fieldtys,\ varenv}$

$$\text{(patfields)}\quad \frac{env \rhd pat_i : ty_i,\ varenv_i \qquad (1 \leq i \leq n)}{env \rhd \{label_1 = pat_1, ..., label_n = pat_n\} : \{label_1 = ty_1, ..., label_n = ty_n\},\ varenv}$$

- where
  - $dom(varenv_i) \ \cap \ dom(varenv_j) \ = \ \emptyset$ for any i, j.
  - $varenv \ = \ varenv_1 \cup \ ... \ \cup varenv_n$

$\boxed{env \rhd (pat, exp) : ty \Rightarrow ty'}$

$$\text{(patexp)}\ \frac{env \rhd pat : ty,\ varenv \quad env + varenv \rhd exp : ty'}{env \rhd (pat, exp) : ty \Rightarrow ty'}$$

$$\text{(patexps)}\ \frac{env \rhd (pat_i, exp_i) : ty \Rightarrow ty' \quad 1 \leq i \leq n}{env \rhd [(pat_1, exp_1),\ \ldots\ , (pat_n, exp_n)] : ty \Rightarrow ty'}$$

$\boxed{env \rhd fieldlist : fieldtys}$

$$\text{(fields)}\ \frac{env \rhd exp_i : ty_i \quad 1 \leq i \leq n}{env \rhd \{label_1 \mapsto exp_1,\ \ldots\ , label_n \mapsto exp_n\} : \{label_1 \mapsto ty_1,\ \ldots\ , label_n \mapsto ty_n\}}$$

2.3.8 Typing declarations

$\boxed{env \rhd decl : (tcenv,\ varenv)}$

$$\text{(val)}\ \frac{env \rhd ty_i \quad env \rhd exp_i : ty_i \quad (1 \leq i \leq n)}{env \rhd VAL\ binds : (\{\},\ varenv)}$$

- where
  - $binds\ =\ [(varinfo_1, exp_1), ..., (varinfo_n, exp_n)]$
  - $varinfo_i\ =\ \{name = id_l,\ ty = ty_i\}$
  - $varenv\ =\ \{id_1 \mapsto VARID\ varinfo_1,\ \ldots\ , id_n \mapsto VARID\ varinfo_n\}$

$$\text{(valrec)}\ \frac{env \rhd ty_i \quad env + varenv \rhd exp_i : ty_i \quad (1 \leq i \leq n)}{env \rhd VALREC\ recbinds : (\{\},\ varenv)}$$

- where
  - $recbinds\ =\ [(varinfo_1, ty_i, exp_1), ..., (varinfo_n, ty_i, exp_n)]$
  - $varinfo_i\ =\ \{name = id_l,\ ty = ty_i\}$
  - $varenv\ =\ \{id_1 \mapsto VARID\ varinfo_1,\ \ldots\ , id_n \mapsto VARID\ varinfo_n\}$

$$\text{(valpolyrec)}\ \frac{env + btvenv \rhd ty_i \quad env + varenv \rhd exp_i : ty_i}{env \rhd VALPOLYREC\ (btvenv,\ polyrecbinds) : (\{\},\ varenv)}$$

- where
  - $polyrecbinds \ = \ [(varinfo_1, ty_1, exp_1), ..., (varinfo_n, ty_n, exp_n)]$
  - $varinfo_i \ = \ \{name = id_l, \ ty = ty_i\}$
  - $varenv \ = \ \{id_1 \mapsto VARID \ varinfo_1, \ ... \ , id_n \mapsto VARID \ varinfo_n\}$

$$\text{(localdec)} \quad \frac{env \rhd decllist_1 : (tcenv_1, \ varenv_1) \qquad env + tcenv_1 + varenv_1 \rhd decllist_2 : (tcenv_2, \ varenv_2)}{env \rhd LOCALDEC \ (decllist_1, decllist_2) : (tcenv_2, \ varenv_2)}$$

$$\text{(datadec)} \quad \frac{}{env \rhd DATADEC \ [tycon_1, ..., tycon_n] : (tcenv, \ \{\})}$$

- where
  - $tcenv \ = \ \{id_1 \mapsto TYCON \ tycon_1, ..., id_n \mapsto TYCON \ tycon_n\}$
  - $tycon_i \ = \ \{name = tyname_i, \ arity = arity_i, \ id = id_i, \ eqkind = eqkind\_ref_i, \ datacon = datacon\_ref_i\}$ for $1 \leq i \leq n$
  - $datacon\_ref_i \ = \ [CONID \ coninfo_1, \ ..., \ CONID \ coninfo_m]$
  - $coninfo_j \ = \ \{name = id_j, \ ty = ty_j, \ funtycon = functycon_j, \ exntag = false, \ tycon = tycon_j\}$ for $1 \leq j \leq n$
  - $eqkind\_ref_i \ = \ EQ$ if all ty_j admit equality. Otherwise, $eqkind\_ref_i \ = \ NONEQ$.
  - the number of bound type vars of $ty_j \ = \ arity_i$
  - $funtycon_j \ = \ true$ if ty_j is a (polymorphic) function type. Otherwise, $funtycon_j \ = \ false$.
  - $tycon_i \ = \ tycon_j$ for all j

$$\text{(datarepdec)} \quad \frac{}{env \rhd DATAREPDEC \ (s, s') : (\{\}, \ \{\})}$$

$$\text{(exndec)} \quad \frac{}{env \rhd EXNDEC \ [coninfo_1, ..., coninfo_n] : (\{\}, \ varenv)}$$

- where
  - $coninfo_i \ = \ \{name = id_i, \ funtycon = funtycon_i \ , \ ty = ty_i, \ exntag = true, \ tycon = tycon_i\}$
  - $funtycon_i = true$ if ty_i is a function type. Otherwise, $funtycon_i = false$.
  - Need to check the well-formedness of $tycon_i$?
  - $varenv \ = \ \{id_1 \mapsto VARID \ \{name = id_1, ty = ty_1\}, ..., id_n \mapsto VARID \ \{name = id_n, ty = ty_n\}\}$

$$\text{(datarepdec)} \ \frac{}{env \rhd EXNREPDEC \ (s, s') : (\{\}, \ \{\})}$$

$$\text{(type)} \ \frac{}{env \rhd TYPE \ tybindinfos : (\{\}, \ \{\})}$$

- $tybindinfos \ = \ [tybindinfo_1, \ ... \ , tybindinfo_n]$
- $tybindinfo_i$ is either $TYCON \ tycon$ or $TYFUN \ \{name : string, tyargs : btvenv, body : ty\}$.
- Need to check the well-formedness of type declarations ?

$\boxed{env \rhd decllist : (tcenv, \ varenv)}$

$$\text{(decllist)} \ \frac{env \rhd decl_i : (tcenv_i, \ varenv_i) \quad (1 \leq i \leq n)}{env \rhd [decl_1, ..., decl_n] : (tcenv_1 + \ ... \ + tcenv_n, \ varenv_1 + \ ... \ + varenv_n)}$$