Appendices

The three appendices constitute self-contained sets of definitions for the three pattern calculi defined in the main body of the dissertation (Appendix C describes the version of first-order implicit typing which uses explicit constraints for structurally polymorphic definitions). The definitions are presented in a formal, structured style suited to reference purposes and include some details which are missing from the main body. Only minimal textual commentary is included at the beginnings of each section and at points where warnings seemed advisable—a full understanding of the motivations and significance of the material should be sought in the text.

The presentations generally follow the inherent order of technical precedence (i.e., kinds before types, types before expressions. etc.), but occasionally revert to a more natural rhetorical order in order to defer less significant details (such as free and bound variable operations) to later sections.

The individual sections of each presentation follow this plan: first meta-variable conventions are noted and atomic and term sorts defined, the latter with context-free grammars; then various auxiliary operations on terms are defined using cases and recursion; and finally, term formation constraints and standard term equivalences are described with formal systems of axioms and inference rules. Although this style is somewhat stilted, it results in presentations which are close to actual implementations in functional languages like Standard ML.

The meta-level notation used throughout the appendices is fairly standard, but we include here a brief key, with emphasis on the more idiosyncratic aspects. A few other local notational conventions are introduced just prior to their (only) use in order to keep their definitions convenient.

Much of the notation is concerned with finite *associations* between atomic symbols and their types or kinds—these are used to track constraints on global constants and free variables. Those used for variables are called *assumptions* and those used for constants (including constructors) are called *settings*. Although we write associations as lists of atom/term pairs and treat them as formal entities when concerned with specifying their formation constraints, we generally use them more informally as functions from atoms to terms in other contexts (e.g., when projecting the types and kinds of constants and variables for the purposes of describing formation of expressions and types). When the range of an association sort is degenerate, i.e., when there are no higher-level terms constraining the variables or constants, the associations are essentially just sets of atomic symbols (we nevertheless make linearity constraints explicit in such cases, as if the association constituted a multi-set). In many cases associations for different sorts and levels are combined as conglomerates—notation is used to effect, e.g., point-wise extension of operations to the components of these conglomerates.

We also use associations to represent substitutions of terms for variables, though we leave the details of substitution over various terms implicit, presuming that it can be easily derived from the definitions of the free variables of terms of that sort.

We keep references to variable assumptions explicit in all the judgement forms, but leave constant and constructor settings implicit, reflecting their static, global nature as parameters to the language definition. It should be remembered that these settings are still subject to certain formation constraints, especially in the case of the higher-order calculus.

Notation

Independence of sets $S \not\cong S'$

$S \not\cong S'$ iff $S \cap S' = \varnothing$

Set difference $S \setminus S'$

as usual in set theory; i.e., $S \setminus S' = \{s \in S \mid s \notin S'\}$

Symbolic equivalence and inequivalence $a \equiv b$, $a \not\equiv b$

predicates presumed given for each atomic sort of symbols

Operational failure **fail**

a special distinguished value; the use of an inference rule is precluded by the failure of any operations used in its antecedents

Indexed tuples and projection $\langle t_1, \ldots, t_n \rangle$; $\cdot_i$

for $t_i \in \mathcal{T}erm$, $1 \leqslant i \leqslant n$, we write $\langle t_1, \ldots, t_n \rangle \in \mathcal{T}erm^n$; when the length is not definite, we write $\mathcal{T}erm^*$

$\langle t_1, \ldots, t_n \rangle_i = t_i$

Tuple length $|\langle t_1, \ldots, t_n \rangle|$

$|\langle t_1, \ldots, t_n \rangle| = n$

Labelled tuples $\langle t_{L_1}, \ldots, t_{L_n} \rangle$; $\cdot_{L_i}$

we also use atomic symbols called *labels* as indices

$\langle t_{L_1}, \ldots, t_{L_n} \rangle_{L_i} = t_i$

Operations extended to tuples $T \odot_L T'$, $T \odot T'$

$\langle L_1{:}t_1, \ldots, L_n{:}t_n \rangle \odot_{L_i} t' =$

$\langle L_1{:}t_1, \ldots, L_i{:}t_i \odot t', \ldots, L_n{:}t_n \rangle$

$\langle L_1{:}A_1, \ldots, L_n{:}A_n \rangle \odot \langle L_1{:}B_1, \ldots, L_n{:}B_n \rangle =$

$\langle L_1{:}A_1 \odot B_1, \ldots, L_n{:}A_n \odot B_n \rangle$

Syntax for associations $\{x_1 \cdot t_1, \ldots, x_n \cdot t_n\}$

for x_i atoms and t_i terms, with the x_i unique; we use various symbols in place of "$\cdot$" for different sorts of associations

Projection from associations $A(x)$

$\{x_1 \cdot t_1, \ldots, x_n \cdot t_n\}(y) =$

if $y \equiv x_i$ for some $1 \leqslant i \leqslant n$ then t_i else **fail**

Domain and range of associations $\mathrm{dom}(A)$, $\mathrm{rng}(A)$

$\mathrm{dom}(\{x_1 \cdot t_1, \ldots, x_n \cdot t_n\}) = \{x_1, \ldots, x_n\}$

$\mathrm{rng}(\{x_1 \cdot t_1, \ldots, x_n \cdot t_n\}) = \{t_1, \ldots, t_n\}$

Merge of associations $A + A'$

$A + A' = \{x \cdot t \mid A'(x) = t \text{ or } A(x) = t \text{ and } x \notin \mathrm{dom}(A')\}$

{NB: the assymmetry in the definition prefers A′ to A}

Weak merge of associations $A \oplus A'$

$A \oplus A' =$ if $\mathrm{dom}(A) \not\cong \mathrm{dom}(A')$ then $A + A'$ else **fail**

Linearity of associations A linear

$\{x_1, \ldots, x_n\}$ linear iff $x_i \neq x_j$ for all $1 \leqslant i \neq j \leqslant n$

Simple substitution $t[s/x]$

substitution in term t of term s for all free occurrences of variable x ; i.e., respecting all binding constructs of t

Substitution for associations $t[A]$

substitution operation extended to an association A sending variables in term t to corresponding range terms

Appendix A: Definitions for the Simply-typed Pattern Calculus

The simply-typed calculus is organized into two "levels", those of *values* and *types*—terms are built up from atomic forms, such as variables and constants, using algebraic and variable-binding constructs. The following diagram illustrates the formation relation between the two levels:

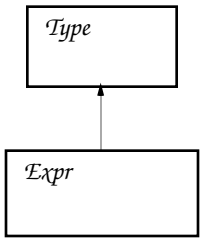

Figure 1: The two levels of the calculus and their relationship

The level of values includes as well a secondary sort of *patterns*. Atomic sorts include proper constants, constructors and variables; associations between atomic value symbols and their types are used to record information about global constants or free variables—these associations are called *assumptions* for variables and *settings* for constants and constructors. Although we represent them formally as lists of atom/term pairs, we will informally think of them as functions from atoms to terms. The additional fine structure of patterns, atomic symbols and corresponding associations is illustrated in figure 2. The overall structure of the presentation is as follows:

I. Types and Value Expressions
II. Patterns
III. Matching and Reduction
IV. Effective Treatment of Patterns

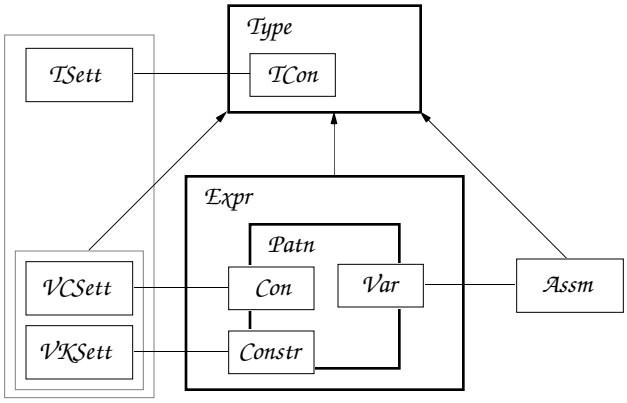

Figure 2: Fine structure of terms, patterns, atomic sorts, and associations in the simply-typed pattern calculus

I. Types and Value Expressions

Types are used to control the formation of value expressions, while the syntax of types themselves is essentially context-free. Types appear explicitly in pattern abstractions as constraints on variables and also in the judgment form for well-typing of expressions. Value expressions represent the actual ground values and functions used to express computations—only expressions which have a valid well-typing are considered sensible. Settings are algebraic signatures which provide types for constants and value constructors, the initial algebraic operators from which ground values are built.

A. Term sorts and syntax

Basic type constants: $B \in \mathcal{TCon}$

Type settings: $C_T \in \mathcal{TSett}$

$C_T ::= \{B, \ldots\}$

Types:

$\tau, \sigma \in \mathcal{Type}$

$\tau ::= 1 \mid B \mid \tau \times \tau \mid \tau \to \tau$

Value constants $c \in \mathcal{Con}$

Value constant settings: $C_{VC} \in \mathcal{VCSett}$

$C_{VC} ::= \{c{:}\tau, \ldots\}$

Standard setting: $C_{VC_0} \in \mathcal{VCSett}$

$C_{VC_0} \equiv \{*{:}1\}$

Value constructor settings: $C_{VK} \in \mathcal{VKSett}$

$C_{VK} ::= \{k{:}\tau \to B, \ldots\}$

Combined settings: $C \in \mathcal{Sett}$

$C ::= \langle C_T, \langle C_{VC}, C_{VK}\rangle\rangle$

Value variables $x, y \in \mathcal{Var}$

Value assumptions: $A \in \mathcal{Assm}$

$A ::= \{x{:}\tau, \ldots\}$

Value expressions: $e, d \in \mathcal{Expr}$

$e ::= c \mid x \mid k \mid e,e \mid (p \Rightarrow e)^+ \mid e\ e \mid \mu x{:}\tau.e$

B. Auxiliary operations *(none).*

C. Inference rules

Several operations and relations on patterns used in the two pattern-abstraction rules are defined in later sections.

Expression formation: $A \vdash e : \tau$

[con] $A \vdash c : C_{VC}(c)$

[constr] $A \vdash k : C_{VK}(k)$

[var] $A \vdash x : A(x)$

[pair] $$\frac{A \vdash e : \tau \quad A \vdash d : \sigma}{A \vdash (e,d) : \tau \times \sigma}$$

$$[abs]\quad \frac{\begin{array}{c}\{p_1, \ldots, p_n\}\ \text{compatible} \\ \{p_1, \ldots, p_n\} \succ \tau \\ \mathcal{A}(p_i) \vdash \mathcal{E}(p_i) : \tau \quad (1 \leqslant i \leqslant n) \\ A + \mathcal{A}(p_i) \vdash e_i : \sigma \quad (1 \leqslant i \leqslant n)\end{array}}{A \vdash (p_1 \Rightarrow e_1 \mid \cdots \mid p_n \Rightarrow e_n) : \tau \to \sigma}$$

$$[app]\quad \frac{A \vdash f : \sigma \to \tau \quad A \vdash e : \sigma}{A \vdash (f\ e) : \tau}$$

$$[rec]\quad \frac{A + \{x{:}\tau\} \vdash e : \tau}{A \vdash (\mu x{:}\tau . e) : \tau}$$

II. Patterns

Patterns are used to define functions; their syntax essentially mirrors that of expressions, but includes explicit constraints on variables and limits function application to constructors. The matching operation on patterns should be intuitively clear, though its definition is deferred until section III. Although we define several relations here in a non-effective way, we will give effective definitions in section IV.

A. Term sorts and syntax

Patterns: $p, q \in \mathcal{P}atn$

$p ::= c \mid x : \tau \mid p\ ,\ p \mid k\ p$

B. Auxiliary operations and relations

Pattern subsumptions: $p \sqsubseteq e,\ p \sqsubseteq q,\ p \sqsubset q$

$p \sqsubseteq e$ iff $\mathcal{M}(p,e) \neq$ **fail**

$p \sqsubseteq q$ iff $p \sqsubseteq \mathcal{E}(q)$

$p \sqsubset q$ iff $p \sqsubseteq q$ and $\neg q \sqsubseteq p$

Pattern coverage: $\{p,\ldots\} \succ \tau$

$\{p_1, \ldots, p_m\} \succ \tau$ iff for all e such that $\vdash e : \tau$, there is an i such that $p_i \sqsubseteq e$

Pattern assumptions: $\mathcal{A} \in \mathcal{P}atn \to \mathcal{A}ssm$

$\mathcal{A}(c) = \{\}$

$\mathcal{A}(x{:}\tau) = \{x{:}\tau\}$

$\mathcal{A}(k\ p) = \mathcal{A}(p)$

$\mathcal{A}(p,q) = \mathcal{A}(p) \oplus \mathcal{A}(q)$

Pattern erasure: $\mathcal{E} \in \mathcal{P}atn \to \mathcal{E}xpr$

$\mathcal{E}(c) = c$

$\mathcal{E}(x{:}\tau) = x$

$\mathcal{E}(k\ p) = k\ (\mathcal{E}(p))$

$\mathcal{E}(p,q) = \mathcal{E}(p)\ ,\ \mathcal{E}(q)$

Pattern overlap and independence: $p \approx q,\ p \not\approx q$

$p \approx q$ iff there is an e such that both $p \sqsubseteq e$ and $q \sqsubseteq e$

$p \not\approx q$ iff $\neg(p \approx q)$

Pattern coherence: $\{p_1, \ldots, p_n\}$ coherent

$\{p_1, \ldots, p_n\}$ coherent iff for all e of the same type,

there is some i, $1 \leqslant i \leqslant n$, such that for all j, $1 \leqslant j \leqslant n$, either $j=i$ or $\neg(p_j \sqsubseteq e)$ or $p_j \sqsubset p_i$

C. Inference rules

Pattern compatibility: $\{p_1, \ldots, p_n\}$ compatible

Use only one of these rules, based on matching regimen.

[first] $\{p_1, \ldots, p_n\}$ compatible

$$[only]\quad \frac{p_i \not\approx p_j \quad (1 \leqslant i \neq j \leqslant n)}{\{p_1, \ldots, p_n\}\ \text{compatible}}$$

$$[best]\quad \frac{\{p_1, \ldots, p_n\}\ \text{coherent}}{\{p_1, \ldots, p_n\}\ \text{compatible}}$$

III. Matching and Reduction

Matching and reduction provide the operational semantics for the calculus. Matching compares a pattern and a term (of the corresponding sort) and returns a substitution of portions of the term for variables in the pattern. The usual beta and eta reduction rules are modified to use matching rather than simple variable substitution—they also require more complex free variable conditions.

A. Term sorts and syntax

Substitutions: $S, R \in \mathcal{S}ubn$

$S ::= [e/x, \ldots]$

B. Auxiliary operations

Free variables of an expression: $FV \in \mathcal{E}xpr \to \wp(\mathcal{V}ar)$

$FV(c) = \{\}$

$FV(x) = \{x\}$

$FV(k) = \{\}$

$FV(e,d) = FV(e) \cup FV(d)$

$FV(p_1 \Rightarrow e_1 \mid \ldots \mid p_n \Rightarrow e_n) = \bigcup_{1 \leqslant i \leqslant n} FV(e_i) \setminus FV(p_i)$

$FV(f\ e) = FV(f) \cup FV(e)$

$FV(\mu x{:}\tau . e) = FV(e) \setminus \{x\}$

Free variables of a pattern: $FV \in \mathcal{P}atn \to \wp(\mathcal{V}ar)$

$FV(p) = FV(\mathcal{E}(p))$

Pattern matching: $\mathcal{M}(p,e) \in \mathcal{P}atn \times \mathcal{E}xpr \to \mathcal{S}ubn$

$\mathcal{M}(c,c) = []$

$\mathcal{M}(x{:}\tau,e) = [e/x]$

$\mathcal{M}(k\ p,\ k\ e) = \mathcal{M}(p,e)$

$\mathcal{M}((p,q),(d,e)) = \mathcal{M}(p,d) + \mathcal{M}(q,e)$

$\mathcal{M}(p,e) =$ **fail**, otherwise

C. Inference rules

In this section we use μ meta-linguistically as the quantifier "the least … such that …"; ι is Russell's *definite descriptor*, "the unique … such that …"}.

Expression conversion: $e = e'$

[beta] $(p_1 \Rightarrow e_1 \mid \cdots \mid p_n \Rightarrow e_n)\ d = e_i\ [\mathcal{M}(p_i,d)]$

where: $i = \mu_j.\ p_j \sqsubseteq d$ *(first-match)*

$p_i = \iota p_j.\ p_j \sqsubseteq d$ *(only-match)*

$p_i = \mu p_j.\ p_j \sqsubseteq d$ *(best-match)*

[eta] $$\frac{FV(p_i) \cong FV(f) \qquad (1 \leqslant i \leqslant n)}{(p_1 \Rightarrow f\ \mathcal{E}(p_1) \mid \cdots \mid p_n \Rightarrow f\ \mathcal{E}(p_n)) = f}$$

We leave implicit rules for reflexive, transitive, symmetric and congruence closure which ensure that = is an equivalence relation on terms.

IV. Effective Treatment of Patterns

In section II above we gave rules for the formation of patterns which were essentially non-effective (e.g., references to "all expressions e of type τ"). Since compilers and type-checkers in a real implementation must be based on finite methods, we present in this section an effective treatment of patterns. The approach is based on the observation that in all cases above where we make reference to an infinite set of patterns or expressions, we can soundly consider just a finite approximation whose elements are bounded in size by the depth of other terms appearing in the context of the reference. For pattern overlap, we give an effective operation such that $p \approx q$ iff $O(p,q)$.

A. Term sorts and syntax

See syntax for patterns in section II above.

B. Auxiliary operations

Pattern depth: $\mathcal{D} \in \mathcal{P}atn \to \mathcal{N}at$

$\mathcal{D}(c) = 0$

$\mathcal{D}(x{:}\tau) = 0$

$\mathcal{D}(k\ p) = 1 + \mathcal{D}(p)$

$\mathcal{D}(p,q) = 1 + \max(\mathcal{D}(p), \mathcal{D}(q))$

Generated patterns: $\mathcal{G}^n \in \mathcal{T}ype \to \wp(\mathcal{P}atn)$

$\mathcal{G}^0(\tau) = \{x{:}\tau\}$ (fresh x)

$\mathcal{G}^{n+1}(1) = \{*\}$

$\mathcal{G}^{n+1}(B) = \{k\ p \mid C_{VK}(k) = \tau \to B,\ p \in \mathcal{G}^n(\tau)\}$

$\mathcal{G}^{n+1}(\tau \times \sigma) = \{p,q \mid p \in \mathcal{G}^n(\tau),\ q \in \mathcal{G}^n(\sigma)\}$

$\mathcal{G}^{n+1}(\tau \to \sigma) = \{x{:}\tau \to \sigma\}$

Pattern coverage: $\{p,\ldots\} \succ \tau$

$\{p_1, \ldots, p_m\} \succ \tau$ iff

for all $q \in \mathcal{G}^n(\tau)$, some $p_i \sqsubseteq q$,

where $n = \max\{\mathcal{D}(p_i) \mid 1 \leqslant i \leqslant m\}$

Pattern overlap: $O \in \mathcal{P}atn \times \mathcal{P}atn \to \mathcal{B}ool$

$O(c,c) = \text{true}$

$O(x{:}\tau, q) = \text{true}$

$O(p, y{:}\sigma) = \text{true}$

$O(k\ p,\ k\ q) = O(p,q)$ *{NB: same* k*}*

$O((p,p'),(q,q')) = O(p,q)$ and $O(p',q')$

$O(p,q) = \text{false, otherwise}$

Appendix B: Definitions for the Higher-order Pattern Calculus

The higher-order pattern calculus is organized into three "levels", those of *values*, *types*, and *kinds*—at each level, terms are built up from atomic forms, such as variables and constants, using various algebraic and variable-binding constructs. The sorts (i.e., sets of terms) at the different levels and the formation relations between them also reveal a finer structure of "subsorts": certain basic expressions denote *ground values* with no active computational import. Ground expressions have as their types a special subset called *data types*, which in turn have as their kinds a special constant D; more general types (including functional and polymorphic types) have as their kind the constant T. In addition, the type level encompasses *type functionals*, e.g., pairs and functions over types, etc., which are not the types of any expression, but rather are used to express various combinations of types. Type functionals in general have more complex kinds involving the product and function constructs for kinds.

The following diagram illustrates the relations between the sorts and subsorts of the three levels, with the vertical arrows indicating the formation relations (i.e., a value term having a certain type or a type term having a certain kind):

In addition to the primary sorts at each level (i.e., expressions, types and kinds), we also have secondary *pattern sorts* at the levels of expressions and types: patterns are used as formal parameters, taking the forms of (a restricted set of) the terms at the same level, against which they are matched, yielding substitutions for the variables they contain (this generalizes the usual single-variable lambda abstraction). Distinct atomic sorts of variables are used at each of the three levels and a special sort of variables called *skolem indices* is used at all levels; atomic constants are used at each of the type and value levels. Associations between atomic symbols and their types or kinds are used to track constraints on global constants and free variables: these associations are called *assumptions* for variable and *settings* for constants. Although we write associations as lists of atom/term pairs, we use them informally as functions from atoms to terms.

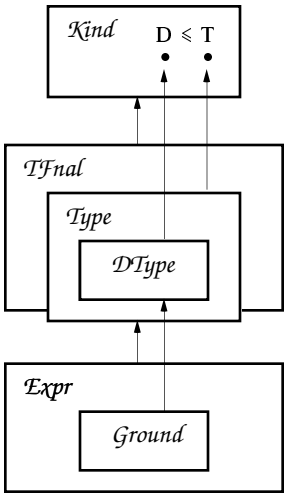

Figure 1: The three levels of the calculus and their formation relations

The additional fine structure of patterns, atomic symbols and corresponding associations is illustrated in figure 2.

The overall structure of the presentation follows this outline:

I. Kinds
II. Types and Type Functionals
III. Value Expressions
IV. Value Patterns
V. Type Patterns
VI. Matching and Conversion
VII. Free Variables
VIII. Effective Treatment of Patterns

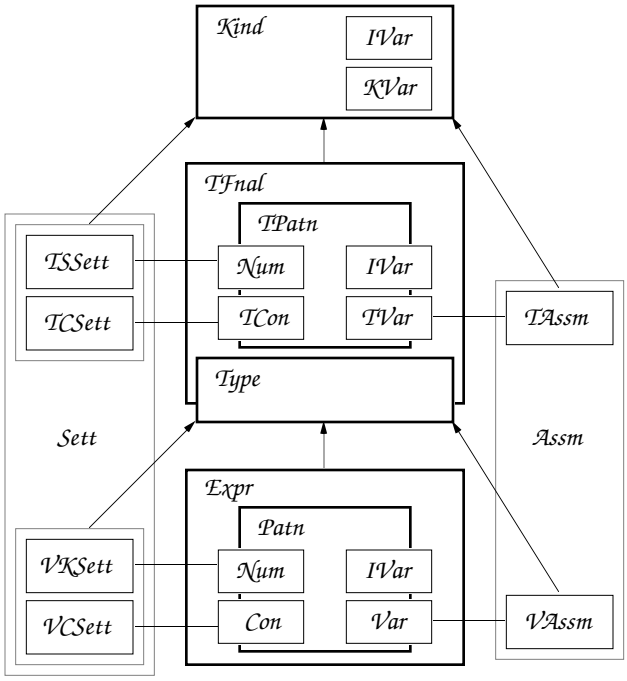

Figure 2: Fine structure of terms, patterns, atomic sorts, and associations in the higher-order calculus

I. Kinds

Kinds are terms used to control the formation of types; that is, they serve essentially as the "types of types"; later we will use certain variations on kinds to enforce various restrictions.

A. Term sorts and syntax

Index variables $\nu, \mu \in \mathcal{IVar}$

Numerals (natural constants) $n, m \in \mathcal{Num}$

$n ::= 0 \mid 1 \mid 2 \mid \cdots$

Indices: $i, j \in \mathcal{Index}$

$i ::= n \mid \nu$

Kind variables $V, X \in \mathcal{KVar}$

Basic kinds: $B \in \mathcal{Basic}$

$B ::= D \mid T$

Proper kinds: $\kappa \in \mathcal{Kind} \subseteq \mathcal{DKind}$

$\kappa ::= 1 \mid T \mid V \mid \mathbf{V}_i \mid \kappa \times \kappa \mid \kappa \to \kappa$

Data kinds: $\delta, \eta \in \mathcal{DKind}$

$\delta ::= 1 \mid D \mid T \mid V \mid \mathbf{V}_i \mid \delta\times\delta \mid \kappa\to\delta$

B. Auxiliary operations

Retractions on data kinds: $\cdot^+, \cdot^- \in \mathcal{DKind}\to\mathcal{DKind}$

$1^+ = 1$ $\quad 1^- = 1$

$D^+ = T$ $\quad D^- = D$

$T^+ = T$ $\quad T^- = D$

$V^+ = V$ $\quad V^- = V$

$\mathbf{V}_i^+ = \mathbf{V}_i$ $\quad \mathbf{V}_i^- = \mathbf{V}_i$

$(\delta\times\eta)^+ = \delta^+ \times \eta^+$ $\quad (\delta\times\eta)^- = \delta^- \times \eta^-$

$(\kappa\to\delta)^+ = \kappa\to(\delta^+)$ $\quad (\kappa\to\delta)^- = \kappa\to(\delta^-)$

Numeral occurrences: $NO \in \mathcal{DKind}\to\wp(\mathcal{Num})$

$NO(\mathbf{V}_n) = \{n\}$

$NO(\mathbf{V}_v) = \{\}$

$NO(\delta\times\eta) = NO(\delta) \cup NO(\eta)$

$NO(\kappa\to\delta) = NO(\delta) \cup NO(\eta)$

$NO(\delta) = \{\}$, otherwise

C. Inference rules

Data kind formation: $\vdash \delta$ dkind

[dkind] $$\frac{n \leqslant |C_{TS}| \quad (n \in NO(\delta))}{\vdash \delta \text{ dkind}}$$

Kind coercion: $\delta \leqslant \eta$

[base] $D \leqslant T$

[prod] $$\frac{\delta \leqslant \delta' \quad \eta \leqslant \eta'}{(\delta\times\eta) \leqslant (\delta'\times\eta')}$$

[fun] $$\frac{\delta \leqslant \delta'}{(\kappa\to\delta) \leqslant (\kappa\to\delta')}$$

[refl] $\delta \leqslant \delta$

Kind conversion: $\delta = \eta$

[sk-inst] $\mathbf{V}_n = C_{TS}(n)$

Plus rules for reflexive, transitive and congruent closure.

II. Types and Type Functionals

Types are used to restrict the formation of value expressions; type functionals, more generally, include pairs of types, functions from types to types, etc.; their formation is restricted in turn by kinds. Data kinds allow us to classify type functionals more finely, as regards uses of vertain constructs.

A. Term sorts and syntax

Type constants $\chi \in \mathcal{TCon}$

Type-level settings: $C_T \in \mathcal{TSett}$

$C_T ::= \langle\{\chi{:}\delta, \ldots\}, \langle\kappa\to B, \ldots\rangle\rangle \equiv \langle C_{TC}, C_{TS}\rangle$

Standard type constant setting: $C_{TC_0} \in \mathcal{TSett}$

$C_{TC_0} \equiv \{*{:}1,\ 1{:}D,\ \times{:}T\times T\to D,\ \to{:}T\times T\to T\}$

Type variables $\alpha, \beta, \varepsilon, \varphi, \psi \in \mathcal{TVar}$

Type-level assumptions: $A_T \in \mathcal{TAssm}$

$A_T ::= \{\alpha{:}\delta, \ldots\}$

Type parameter patterns: $\pi,\upsilon \in \mathcal{TParm} \subseteq \mathcal{TPatn}$

$\pi ::= * \mid \alpha : \delta \mid \pi\ ,\ \pi$

Type functionals: $\theta, \omega \in \mathcal{TFnal}$

$\theta ::= \chi \mid \alpha \mid \mathbf{S}_i \mid \mathbf{T}_{ij} \mid \theta,\theta \mid \theta\ \theta \mid \lambda\pi.\theta \mid \forall\pi.\theta \mid \forall V.\theta$

Types proper: $\tau, \sigma \in \mathcal{Type}$

$\mathcal{Type} = \{\theta \mid A_T \vdash \theta : T,$ for some (appropriate) $A_T\}$

{see below for the type formation judgement}

B. Auxiliary operations

Retraction for type assumptions: $\cdot^- \in \mathcal{TAssm}\to\mathcal{TAssm}$

$\{\alpha_1{:}\delta_1, \ldots, \alpha_n{:}\delta_n\} = \{\alpha_1{:}\delta_1{}^-, \ldots, \alpha_n{:}\delta_n{}^-\}$

Parameter assumptions: $\mathcal{A} \in \mathcal{TParm}\to\mathcal{TAssm}$

$\mathcal{A}(*) = \{\}$

$\mathcal{A}(\alpha{:}\delta) = \{\alpha{:}\delta\}$

$\mathcal{A}(\pi,\upsilon) = \mathcal{A}(\pi) \oplus \mathcal{A}(\upsilon)$

{NB: this is equivalent to $\mathcal{A}(\pi)$ defined below for type patterns in general}

Contingent retraction: $\cdot/\cdot \in \mathcal{DKind}\times\mathcal{DKind}\to\mathcal{DKind}$

$\delta/\kappa = \textbf{if } (\kappa=\kappa^-) \textbf{ then } \delta \textbf{ else } \delta^+$

C. Inference rules

For the scope of this section, we abbreviate $A \triangleq A_T$. The free variable operations FV are defined in section VII.

Constant setting formation: $\vdash C_{TC}$ tcsett

[tcsett] $$\frac{\vdash \delta_1 \text{ dkind} \quad \cdots \quad \vdash \delta_n \text{ dkind}}{\vdash \{\chi_1{:}\delta_1, \ldots, \chi_n{:}\delta_n\} \text{ tcsett}}$$

Constructor setting formation: $\vdash C_{TS}$ tssett

[tssett] $$\frac{\vdash \kappa_1 \text{ dkind} \quad \cdots \quad \vdash \kappa_n \text{ dkind}}{\vdash \langle \kappa_1 \to B_1, \ldots, \kappa_n \to B_n \rangle \text{ tssett}}$$

Combined setting formation: $\vdash C_T$ tsetting

[tsett] $$\frac{\vdash C_{TC} \text{ tcsett} \quad \vdash C_{TS} \text{ tssett}}{\vdash \langle C_{TC}, C_{TS} \rangle \text{ tsetting}}$$

Type assumption formation: $\vdash A_T$ tassm

[tassm] $$\frac{\vdash \delta_1 \text{ dkind} \quad \cdots \quad \vdash \delta_n \text{ dkind}}{\vdash \{\alpha_1{:}\delta_1, \ldots, \alpha_n{:}\delta_n\} \text{ tassm}}$$

Type formation and classification: $A_T \vdash \theta : \delta$

[t-const] $A \vdash \chi : C_{TC}(\chi)$

[t-var] $A \vdash \alpha : A_T(\alpha)$

[cstruc] $A \vdash \mathbf{S}_n : C_{TS}(n)$

[vstruc] $A \vdash \mathbf{S}_v : \mathbf{V}_v \to D$

[ccarg] $$\frac{m \leqslant |C_{VK}|}{A \vdash \mathbf{T}_{nm} : C_{TS}(n)}$$

[cvarg] $A \vdash \mathbf{T}_{n\mu} : C_{TS}(n)$

[vvarg] $A \vdash \mathbf{T}_{v\mu} : \mathbf{V}_v \to D$

{NB: variable primary index and constant secondary is not allowed}

[t-pair] $$\frac{A \vdash \theta : \kappa \quad A \vdash \omega : \eta}{A \vdash (\theta,\omega) : \kappa \times \eta}$$

[t-app] $$\frac{A \vdash \omega : \kappa \to \eta \quad A \vdash \theta : \delta \quad \delta \leqslant \kappa}{A \vdash (\omega\ \theta) : \eta \,/\, \delta}$$

[t-lam] $$\frac{\vdash \pi : \kappa \quad A +_T \mathcal{A}(\pi)^- \vdash \theta : \delta}{A \vdash (\lambda\pi.\theta) : \kappa \to \delta}$$

[all] $$\frac{\vdash \pi : \delta \quad A +_T \mathcal{A}(\pi) \vdash \theta : T}{A \vdash (\forall \pi.\theta) : T}$$

[k-poly] $$\frac{A \vdash \theta : T}{A \vdash (\forall V.\theta) : T}$$

[coerce] $$\frac{A \vdash \theta : \delta \quad \delta \leqslant \eta}{A \vdash \theta : \eta}$$

[subj-conv] $$\frac{A \vdash \theta : \delta \quad \theta = \theta'}{A \vdash \theta' : \delta}$$

{NB: π-formation is defined below for arbitrary *type patterns, i.e., including the subset of type parameters}*

Type conversion: $\theta = \theta'$

[sk-inst] $\mathbf{T}_{nm} = C_{VK}(n)(m)$

[t-beta] $(\lambda\pi.\theta)\ \omega = \theta\ [\mathcal{M}(\pi,\theta)]$

[t-eta] $$\frac{\omega = \mathcal{E}(\pi) \quad FV(\pi) \ncong FV(\theta)}{(\lambda\pi.\theta\ \omega) = \theta}$$

We need, in addition to these, routine rules for the reflexive, transitive, symmetric and compatible closure; we assume appropriate handling of alpha-congruence.

III. Value Expressions

Value expressions represent the actual ground values and functions used to express computations. Constructor settings are algebraic signatures which provide types for value constructors, the initial algebraic operators from which ground values are built; as for type settings, we consider them global parameters of the language definition.

A. Term sorts and syntax

Value constants $c \in \mathcal{C}on$

Value constant settings: $C_{VC} \in \mathcal{VCS}ett$

C_{VC} ::= $\{c{:}\theta, \ldots\}$

Standard setting: $C_{VC_0} \in \mathcal{VCS}ett$

$C_{VC_0} \equiv \{*{:}1\}$

Value constructor settings: $C_{VK} \in \mathcal{VKS}ett$

C_{VK} ::= $\langle\langle\theta, \ldots\rangle, \ldots, \langle\theta, \ldots\rangle\rangle$

Combined settings: $C \in \mathcal{S}ett$

C ::= $\langle C_T, \langle C_{VC}, C_{VK}\rangle\rangle$

Value variables $x, y \in \mathcal{V}ar$

Combined value-level assumptions: $A \in \mathcal{A}ssm$

$A ::= \langle A_T, \{x{:}\theta, \ldots\}\rangle \equiv \langle A_T, A_V\rangle$

Value expressions: $e, d \in \mathcal{E}xpr$

$e ::= c \mid x \mid \mathbf{K}_{ij} \mid e,e \mid (p \Rightarrow e)^+ \mid e\ e \mid (\rho \Rightarrow\!\!> e)^+ \mid e{\downarrow}\theta \mid \mu x{:}\theta.e \mid \Lambda V.e \mid e{\downarrow}\kappa$

B. Auxiliary operations *(none).*

C. Inference rules

Several operations and relations on patterns used in the two pattern-abstraction rules are defined in later sections.

Value constant setting formation: $\vdash C_{VC}$ vcsett

$$[vcs]\quad \frac{\{\} \vdash \tau_1 : T \quad \cdots \quad \{\} \vdash \tau_n : T}{\vdash \{c_1{:}\tau_1, \ldots, c_n{:}\tau_n\}\ \text{vcsett}}$$

Value constructor setting formation: $\vdash C_{VK}$ vksett

$$[vks]\quad \frac{\begin{array}{c} n = |C_{TS}| \\ m_1 \geqslant 1 \quad \cdots \quad m_n \geqslant 1 \\ \{\} \vdash \theta_{11} : \delta_{11} \quad \delta_{11} \leqslant C_{TS}(1) \\ \cdots \qquad\qquad \cdots \\ \{\} \vdash \theta_{nm_n} : \delta_{nm_n} \quad \delta_{nm_n} \leqslant C_{TS}(n) \end{array}}{\vdash \langle\langle\theta_{11}, \ldots, \theta_{1m_1}\rangle, \ldots, \langle\theta_{n1}, \ldots, \theta_{nm_n}\rangle\rangle\ \text{vksett}}$$

Combined setting formation: $\vdash C$ setting

$$[sett]\quad \frac{\vdash C_T\ \text{tsett} \quad \vdash C_{VC}\ \text{vcsett} \quad \vdash C_{VK}\ \text{vksett}}{\vdash \langle C_T, \langle C_{VC}, C_{VK}\rangle\rangle\ \text{setting}}$$

Value assumption formation: $\vdash A$ assm

$$[assm]\quad \frac{\vdash A_T\ \text{tassm} \quad A_T \vdash \tau_1 : T \quad \cdots \quad A_T \vdash \tau_n : T}{\vdash \langle A_T, \{x_1{:}\tau_n, \ldots, x_n{:}\tau_n\}\rangle\ \text{assm}}$$

Expression formation: $A \vdash e : \tau$

$$[con]\quad A \vdash c : C_{VC}(c)$$

$$[var]\quad A \vdash x : A_V(x)$$

$$[cc\text{-}con]\quad \frac{C_{TS}(n) = \kappa \to B \qquad (C_{VK}(n)(m))\ \alpha = \theta}{A \vdash \mathbf{K}_{nm} : \forall\alpha{:}\kappa.\ \theta \to \mathbf{S}_n\ \alpha}$$

$$[cv\text{-}con]\quad \frac{C_{TS}(n) = \kappa \to B}{A \vdash \mathbf{K}_{n\mu} : \forall\alpha{:}\kappa.\ \mathbf{T}_{n\mu}\ \alpha \to \mathbf{S}_n\ \alpha}$$

$$[vv\text{-}con]\quad A \vdash \mathbf{K}_{v\mu} : \forall\alpha{:}\mathbf{V}_v.\ \mathbf{T}_{v\mu}\ \alpha \to \mathbf{S}_v\ \alpha$$

$$[pair]\quad \frac{A \vdash e : \tau \qquad A \vdash d : \sigma}{A \vdash (e,d) : \tau \times \sigma}$$

$$[abs]\quad \frac{\begin{array}{c} \{p_1, \ldots, p_n\}\ \text{compatible} \\ \{p_1, \ldots, p_n\} \succ \tau \\ A_T \vdash p_i : \tau \qquad (1 \leqslant i \leqslant n) \\ A + \mathcal{A}(p_i) \vdash e_i : \sigma \qquad (1 \leqslant i \leqslant n) \end{array}}{A \vdash (p_1 \Rightarrow e_1 \mid \cdots \mid p_n \Rightarrow e_n) : \tau \to \sigma}$$

$$[app]\quad \frac{A \vdash f : \sigma \to \tau \qquad A \vdash e : \sigma}{A \vdash (f\ e) : \tau}$$

$$[rec]\quad \frac{A_T \vdash \tau : T \qquad A +_V \{x{:}\tau\} \vdash e : \tau}{A \vdash (\mu x{:}\tau.e) : \tau}$$

$$[t\text{-}abs]\quad \frac{\begin{array}{c} \{\rho_1, \ldots, \rho_n\}\ \text{compatible} \\ \{\rho_1, \ldots, \rho_n\} \succ \mathcal{K}(\pi) \\ \left(FV(\pi)_T \cup \bigcup_i FV(\rho_i)_T\right) \ncong FV(A)_T \\ \vdash \rho_i : \mathcal{K}(\pi) \qquad (1 \leqslant i \leqslant n) \\ A +_T \mathcal{A}(\rho_i) \vdash e_i : \tau[\mathcal{M}(\pi,\rho_i)] (1 \leqslant i \leqslant n) \end{array}}{A \vdash (\rho_1 \Rightarrow\!\!> e_1 \mid \cdots \mid \rho_n \Rightarrow\!\!> e_n) : \forall\pi.\tau}$$

$$[t\text{-}inst]\quad \frac{A \vdash f : \forall\pi.\tau \qquad A_T \vdash \theta : \mathcal{K}(\pi)}{A \vdash (f{\downarrow}\theta) : \tau\ [\mathcal{M}(\pi,\theta)]}$$

$$[k\text{-}abs]\quad \frac{V \notin FV_{KV}(A) \qquad A \vdash e : \tau}{A \vdash (\Lambda V.e) : \forall V.\tau}$$

$$[k\text{-}inst]\quad \frac{A \vdash f : \forall V.\tau}{A \vdash (f{\downarrow}\kappa) : \tau\ [\kappa/V]}$$

IV. Value Patterns

Value patterns are used to define functions over values; their syntax essentially mirrors that of expressions, but includes explicit constraints on variables and limits function application and type instantiation to constructors. The matching operation on patterns should be intuitively clear, though its definition is deferred until section VI. Although we define several relations here in a non-effective way, we will give effective definitions in section VIII.

A. Term sorts and syntax

Value patterns: $p, q \in \mathcal{P}atn$

$p ::= c \mid x : \tau \mid p , p \mid \mathbf{K}_{ij}{\downarrow}\theta\ p$

B. Auxiliary operations and relations

Pattern subsumptions: $p \sqsubseteq e,\ p \sqsubseteq q,\ p \sqsubset q$

$p \sqsubseteq e$ iff $\mathcal{M}(p,e) \neq \mathbf{fail}$

$p \sqsubseteq q$ iff $p \sqsubseteq \mathcal{E}(q)$

$p \sqsubset q$ iff $p \sqsubseteq q$ and $\neg\, q \sqsubseteq p$

Pattern coverage: $\{p,\ldots\} \succ \tau$

$\{p_1, \ldots, p_m\} \succ \tau$ iff for all e such that $\vdash e{:}\tau$, there is an i such that $p_i \sqsubseteq e$

Pattern assumptions: $\mathcal{A} \in \mathcal{P}atn \to \mathcal{V}\mathcal{A}ssm$

$\mathcal{A}(c) = \{\}$

$\mathcal{A}(x{:}\tau) = \{x{:}\tau\}$

$\mathcal{A}(\mathbf{K}_{ij}{\downarrow}\theta\ p) = \mathcal{A}(p)$

$\mathcal{A}(p,q) = \mathcal{A}(p) \oplus \mathcal{A}(q)$

Type constraints of a pattern: $\mathcal{C} \in \mathcal{P}atn \to \wp(\mathcal{T}\mathcal{F}nal)$

$\mathcal{C}(p) = \mathrm{rng}(\mathcal{A}(p))$

Pattern erasure: $\mathcal{E} \in \mathcal{P}atn \to \mathcal{E}xpr$

$\mathcal{E}(c) = c$

$\mathcal{E}(x{:}\tau) = x$

$\mathcal{E}(\mathbf{K}_{ij}{\downarrow}\theta\ p) = \mathbf{K}_{ij}{\downarrow}\theta\ (\mathcal{E}(p))$

$\mathcal{E}(p,q) = \mathcal{E}(p) , \mathcal{E}(q)$

Pattern overlap and independence: $p \approx q,\ p \not\approx q$

$p \approx q$ iff there is an e such that both $p \sqsubseteq e$ and $q \sqsubseteq e$

$p \not\approx q$ iff $\neg(p \approx q)$

Pattern coherence: $\{p_1, \ldots, p_n\}$ coherent

$\{p_1, \ldots, p_n\}$ coherent iff for all e of the same type,

there is some i, $1 \leqslant i \leqslant n$, such that for all j, $1 \leqslant j \leqslant n$, either $j=i$ or $\neg(p_j \sqsubseteq e)$ or $p_j \sqsubset p_i$

C. Inference rules

Pattern compatibility: $\{p_1, \ldots, p_n\}$ compatible

Use only one of these rules, based on matching regimen.

[first] $\{p_1, \ldots, p_n\}$ compatible

[only]
$$\frac{p_i \not\approx p_j \qquad (1 \leqslant i \neq j \leqslant n)}{\{p_1, \ldots, p_n\} \text{ compatible}}$$

[best]
$$\frac{\{p_1, \ldots, p_n\} \text{ coherent}}{\{p_1, \ldots, p_n\} \text{ compatible}}$$

Pattern formation: $A_T \vdash p : \tau$

[patn]
$$\frac{A_T \vdash \sigma : T \qquad (\sigma \in \mathcal{C}(p)) \qquad \langle A_T, \mathcal{A}(p)\rangle \vdash \mathcal{E}(p) : \tau}{A_T \vdash p : \tau}$$

V. Type Patterns

As patterns are used to define functions, so type patterns are used to define polymorphic values; correspondingly, the syntax of type patterns mirrors that of types and type functionals. At the level of types, however, we must allow for patterns which range over higher-order type functions: this complicates the syntax by forcing distinctions between lambda-bound and pattern-bound variables. Flat parameter patterns are used to avoid bilateral substitutions in pattern matching. Once again, effective versions of coverage and coherence are deferred until section VIII.

A. Term sorts and syntax

Flat kinds: $\kappa^\circ \in \mathcal{K}ind^\circ \subseteq \mathcal{K}ind$

$\kappa^\circ ::= 1 \mid T \mid V \mid \mathbf{V}_i \mid \kappa \to \kappa^\circ$

Flat type parameter patterns: $\pi^\circ, \upsilon^\circ \in \mathcal{T}\mathcal{P}arm^\circ \subseteq \mathcal{T}\mathcal{P}arm$

$\pi^\circ ::= * \mid \alpha : \kappa^\circ \mid \pi^\circ , \pi^\circ$

Atomic type patterns: $\Phi, \Theta \in \mathcal{A}t\mathcal{P}atn$

$\Phi ::= \alpha \mid \chi \mid \mathbf{S}_i$

Type patterns: $\rho, \upsilon \in \mathcal{T}\mathcal{P}atn$

$\rho ::= \rho,\rho \mid \Phi\ \rho \cdots \rho \mid \lambda\pi^\circ.\rho \mid \psi\ \rho \cdots \rho : \delta$

B. Auxiliary operations and relations

Type pattern erasure: $\mathcal{E} \in \mathcal{T}\mathcal{P}atn \to \mathcal{T}\mathcal{F}nal$

$\mathcal{E}(\rho,\rho') = \mathcal{E}(\rho) , \mathcal{E}(\rho')$

$\mathcal{E}(\Phi\ \rho_1 \cdots \rho_n) = \Phi\ \mathcal{E}(\rho_1) \cdots \mathcal{E}(\rho_n)$

$\mathcal{E}(\lambda\pi^\circ.\rho) = \lambda\pi^\circ.\ \mathcal{E}(\rho)$

$\mathcal{E}(\psi\ \rho_1 \cdots \rho_n : \delta) = \psi\ \mathcal{E}(\rho_1) \cdots \mathcal{E}(\rho_n)$

Kind projection: $\mathcal{K} \in \mathcal{T}\mathcal{P}arm \to \mathcal{D}\mathcal{K}ind$

$\mathcal{K}(*) = 1$

$\mathcal{K}(\alpha{:}\delta) = \delta$

$\mathcal{K}(\pi,\nu) = \mathcal{K}(\pi) \times \mathcal{K}(\nu)$

Tupled arrow notation: $\langle\kappa,\ldots\rangle \to \delta$

$\langle\rangle \to \delta \triangleq \delta$

$\langle\kappa_1, \ldots, \kappa_n\rangle \to \delta \triangleq \kappa_1 \to (\cdots(\kappa_n \to \delta)\ \cdots)$

Domain extended to parameter tuples: $\mathrm{dom}(\Pi)$

$\mathrm{dom}(\langle\pi_1, \ldots, \pi_n\rangle) = \mathrm{dom}(\mathcal{A}(\pi_1) \oplus \cdots \oplus \mathcal{A}(\pi_n))$

{NB: success of the operation implies linearity of the π_i}

Extension of functions to tuples $f^*\ T$

$f^*\ \langle t_1, \ldots, t_n\rangle = \langle f\ t_1, \ldots, f\ t_n\rangle$

Core projection: $\mathcal{A} \in \mathcal{T}\mathcal{P}arm^* \to \mathcal{T}\mathcal{P}atn \to \mathcal{T}\mathcal{A}ssm$

$\mathcal{A}_\Pi(\chi) = \{\}$

$\mathcal{A}_\Pi(\alpha) =$ **if** $\alpha \in \mathrm{dom}(\Pi)$ **then** $\{\}$ **else fail**

$\mathcal{A}_\Pi(\mathbf{S}_i) = \{\}$

$\mathcal{A}_\Pi(\rho,\rho') = \mathcal{A}_\Pi(\rho) \oplus \mathcal{A}_\Pi(\rho')$

$\mathcal{A}_\Pi(\Phi\ \rho_1 \cdots \rho_n) = \mathcal{A}_\Pi(\Phi) \oplus \mathcal{A}_\Pi(\rho_1) \oplus \cdots \oplus \mathcal{A}_\Pi(\rho_n)$

$\mathcal{A}_\Pi(\lambda\pi.\rho) = \mathcal{A}_{\Pi,\pi}(\rho)$

$\mathcal{A}_\Pi((\psi\ \rho_1 \cdots \rho_n):\delta) =$

if $\mathcal{E}^*(\Pi) = \langle \rho_1, \ldots, \rho_n \rangle$ **and** $\psi \notin \mathrm{dom}(\Pi)$

then $\{\psi : \mathcal{K}^*(\Pi) \to \delta\}$ **else fail**

Assumption projection proper: $\mathcal{A} \in \mathcal{TP}atn \to \mathcal{TA}ssm$

$\mathcal{A}(\rho) = \mathcal{A}_{\langle\rangle}(\rho)$

Kind constraints of a type pattern: $\mathcal{C} \in \mathcal{TP}atn \to \wp(\mathcal{DK}ind)$

$\mathcal{C}(\rho) = \mathrm{rng}(\mathcal{A}(\rho))$

Type pattern subsumptions: $\rho \sqsubseteq \theta,\ \rho \sqsubseteq \upsilon,\ \rho \sqsubset \upsilon$

$\rho \sqsubseteq \theta$ iff $\mathcal{M}(\rho,\theta) \neq$ **fail**

$\rho \sqsubseteq \upsilon$ iff $\rho \sqsubseteq \mathcal{E}(\upsilon)$

$\rho \sqsubset \upsilon$ iff $\rho \sqsubseteq \upsilon$ and $\neg\, \upsilon \sqsubseteq \rho$

Type pattern coverage: $\{\rho, \ldots\} \succ \delta$

$\{\rho_1, \ldots, \rho_m\} \succ \delta$ iff for all θ such that $\vdash \theta:\delta$, there is an i such that $\rho_i \sqsubseteq \theta$

Type pattern overlap and independence: $\rho \approx \upsilon,\ \rho \not\approx \upsilon$

$\rho \approx \upsilon$ iff there is a θ such that both $(\rho \sqsubseteq \theta)$ and $(\upsilon \sqsubseteq \theta)$

$\rho \not\approx \upsilon$ iff $\neg(\rho \approx \upsilon)$

Type pattern coherence: $\{\rho_1, \ldots, \rho_n\}$ coherent

$\{\rho_1, \ldots, \rho_n\}$ coherent iff for all θ of the same kind,

there is some i, $1 \leqslant i \leqslant n$, such that for all j, $1 \leqslant j \leqslant n$, either $j=i$ or $\neg(\rho_j \sqsubseteq \theta)$ or $\rho_j \sqsubset \rho_i$

C. Inference rules

Type pattern compatibility: $\{\rho_1, \ldots, \rho_n\}$ compatible

Use only one of these rules, based on matching regimen.

[first] $\{\rho_1, \ldots, \rho_n\}$ compatible

[only]
$$\frac{\rho_i \not\approx \rho_j \qquad (1 \leqslant i \neq j \leqslant n)}{\{\rho_1, \ldots, \rho_n\} \text{ compatible}}$$

[best]
$$\frac{\{\rho_1, \ldots, \rho_n\} \text{ coherent}}{\{\rho_1, \ldots, \rho_n\} \text{ compatible}}$$

Type pattern formation: $\vdash \rho : \delta$

[tpatn]
$$\frac{\vdash \eta\ \mathrm{dkind} \qquad (\eta \in \mathcal{C}(\rho)) \qquad \mathcal{A}(\rho) \vdash \mathcal{E}(\rho) : \delta}{\vdash \rho : \delta}$$

VI. Matching and Conversion

Matching and reduction provide the operational semantics for the calculus; conversion is the equivalence generated by reduction. Matching compares a pattern and a term (of the corresponding sort) and returns a substitution of portions of the term for variables in the pattern. The usual beta and eta reduction rules are modified to use matching rather than simple variable substitution—they also require more complex free variable conditions.

A. Term sorts and syntax

Kind-level substitutions: $S_K \in \mathcal{KS}ubn$

$S_{KV} \in \mathcal{KVS}ubn,\ S_{KI} \in \mathcal{IS}ubn$

$S_K ::= \langle [\kappa/V, \ldots], [i/\nu, \ldots] \rangle \equiv \langle S_{KV}, S_{KI} \rangle$

Cumulative type-level substitutions: $S_T \in \mathcal{TS}ubn$

$S_{TV} \in \mathcal{TVS}ubn,\ S_{TI} \in \mathcal{IS}ubn$

$S_T ::= \langle S_K, \langle [\theta/\alpha, \ldots], [i/\nu, \ldots] \rangle \rangle \equiv \langle S_K, \langle S_{TV}, S_{TI} \rangle \rangle$

Combined substitutions: $S, R \in \mathcal{S}ubn$

$S ::= \langle S_T, [e/x, \ldots] \rangle \equiv \langle S_T, S_V \rangle$

Empty substitutions: $\varnothing_K \in \mathcal{KS}ubn;\ \varnothing_T \in \mathcal{TS}ubn;\ \varnothing \in \mathcal{S}ubn$

$\varnothing_K \equiv \langle [], [] \rangle$

$\varnothing_T \equiv \langle \varnothing_K, \langle [], [] \rangle \rangle$

$\varnothing \equiv \langle \varnothing_T, [] \rangle$

B. Auxiliary operations

Composition of substitutions: $S \circ R \in \mathcal{S}ubn \times \mathcal{S}ubn \to \mathcal{S}ubn$

$S \circ [t_1/v_1, \ldots, t_n/v_n] = S + [t_1[S]/v_1, \ldots, t_n[S]/v_n]$

{NB: extended in the obvious way for conglomerates}

Kind matching: $\mathcal{M}(V,\kappa) \in \mathcal{KV}ar \times \mathcal{K}ind \to \mathcal{KS}ubn$

$\mathcal{M}(V,\kappa) = \langle [\kappa/V], [] \rangle$

Index matching: $\mathcal{M}(i,j) \in \mathcal{I}ndex \times \mathcal{I}ndex \to \mathcal{IS}ubn$

$\mathcal{M}(n,m) =$ if $n=m$ then [] else **fail**

$\mathcal{M}(n,\mu) = [n/\mu]$

$\mathcal{M}(\nu,j) = [j/\nu]$

Type pattern matching: $\mathcal{M}(\rho,\theta) \in \mathcal{TP}atn \times \mathcal{TF}nal \to \mathcal{TS}ubn$

{NB: this definition assumes that the type argument has been reduced to (head-) normal form}

$\mathcal{M}(\chi,\chi) = \varnothing_T$

$\mathcal{M}(\alpha,\alpha) = \varnothing_T$ *{NB: same α}*

$\mathcal{M}(\mathbf{S}_i,\mathbf{S}_j) = \langle \langle [], \mathcal{M}(i,j) \rangle, \langle [], [] \rangle \rangle$

$\mathcal{M}((\rho,\rho'),(\theta,\theta')) =$ **let** $S=\mathcal{M}(\rho,\theta)$ **in** $\mathcal{M}(\rho'[S],\ \theta'[S])$

$\mathcal{M}(\Phi\ \rho_1 \cdots \rho_n, \varphi\ \theta_1 \cdots \theta_n) =$

let $S_0=\mathcal{M}(\Phi,\varphi)$ **and**

for i=1 to n,

$R_i=\mathcal{M}(\rho_i[S_{i-1}],\ \theta_i[S_{i-1}])$ **and** $S_i=S_{i-1}\circ R_i$

in S_n

$\mathcal{M}(\lambda\pi.\rho,\lambda\nu.\theta) = \mathcal{M}(\rho[\mathcal{M}(\pi,\nu)],\ \theta)$

$\mathcal{M}((\psi\ \rho_1 \cdots \rho_n):\delta,\theta) = \langle\varnothing_K,\ \langle[(\lambda\rho_1.\ \ldots\ \lambda\rho_n.\theta)/\psi],\ []\rangle\rangle$

{NB: a slight abuse in that the ρ_i are unconstrained}

$\mathcal{M}(\rho,\theta)$ = **fail**, otherwise

Matching for type-pattern pairs: $\mathcal{M} \in \mathcal{TP}atn\times\mathcal{TP}atn\rightarrow\mathcal{S}ubn$

$\mathcal{M}(\rho,\ \upsilon) = \mathcal{M}(\rho,\ \mathcal{E}(\upsilon))$

Value pattern matching: $\mathcal{M}(p,e) \in \mathcal{P}atn\times\mathcal{E}xpr\rightarrow\mathcal{S}ubn$

$\mathcal{M}(c,c) = \varnothing$

$\mathcal{M}(x{:}\tau,e) = \langle\varnothing_T,\ [e/x]\rangle$

$\mathcal{M}(\mathbf{K}_{ij}{\downarrow}\theta\ p,\mathbf{K}_{hk}{\downarrow}\omega\ e) =$

let $S=\langle\langle[],\ \mathcal{M}(i,j)\rangle,\ \langle[],\ \mathcal{M}(k,h)\rangle\rangle,\ []\rangle$

in $\mathcal{M}(p[S],\ e[S])$

$\mathcal{M}((p,q),(d,e)) =$ **let** $S=\mathcal{M}(p,d)$ **in** $\mathcal{M}(q[S],\ e[S])$

$\mathcal{M}(p,e)$ = **fail**, otherwise

{NB: progressive substitution is necessary here for indices, which may be non-linear}

Matching for pattern pairs: $\mathcal{M}(p,\ q) \in \mathcal{P}atn\times\mathcal{P}atn\rightarrow\mathcal{S}ubn$

$\mathcal{M}(p,\ q) = \mathcal{M}(p,\ \mathcal{E}(q))$

C. Inference rules

In this section we use μ meta-linguistically as the quantifier "the least … such that …"; ι is Russell's *definite descriptor*, "the unique … such that …"}.

Expression conversion: $e=e'$

[beta] $(p_1\Rightarrow e_1 \mid \cdots \mid p_n\Rightarrow e_n)\ d = e_i\ [\mathcal{M}(p_i,d)]$

where: $i = \mu_j.\ p_j \sqsubseteq d$ *(first-match)*

$p_i = \iota p_j.\ p_j \sqsubseteq d$ *(only-match)*

$p_i = \mu p_j.\ p_j \sqsubseteq d$ *(best-match)*

[eta] $$\frac{FV(p_1)\ncong FV(f) \quad \ldots \quad FV(p_n)\ncong FV(f)}{(p_1 \Rightarrow f\ \mathcal{E}(p_1) \mid \cdots \mid p_n \Rightarrow f\ \mathcal{E}(p_n)) = f}$$

[t-beta] $(\rho_1\Rightarrow e_1 \mid \cdots \mid \rho_n\Rightarrow e_n){\downarrow}\theta = e_i\ [\mathcal{M}(\rho_i,\theta)]$

where: $i = \mu j.\ \rho_j \sqsubseteq \theta$ *(first-match)*

$\rho_i = \iota\rho_j.\ \rho_j \sqsubseteq \theta$ *(only-match)*

$\rho_i = \mu\rho_j.\ \rho_j \sqsubseteq \theta$ *(best-match)*

$$\textit{[t-eta]}\ \frac{FV(\rho_1)\ncong FV(f) \quad \ldots \quad FV(\rho_n)\ncong FV(f)}{(\rho_1 \twoheadrightarrow f{\downarrow}\mathcal{E}(\rho_1) \mid \cdots \mid \rho_n \twoheadrightarrow f{\downarrow}\mathcal{E}(\rho_n)) = f}$$

[mu] $\mu x{:}\theta.e = e\ [(\mu x{:}\theta.e)/x]$

We leave implicit rules for reflexive, transitive, symmetric and congruence closure which ensure that = is an equivalence relation on terms; note that we must also ensure that expression conversion is congruent with respect to type conversion. Finally, in order to preserve confluence properties in the cases of the first- and best-fit regimens we must require that argument expressions are at least as refined as all patterns in the expression, in the sense of the ordering $\sqsubseteq$, before allowing application of either form of β reduction.

VII. Free Variables

Because of the complex syntax of patterns and our use of several sorts of variables (type variables, skolem indices, etc.), we need more extensive bookkeeping operations for variables than the usual notion of free variables of a term.

A. Term sorts and syntax

Kind-level variable collections: $VC_K \in \mathcal{KVC}$

$VC_K ::= \langle\{V, \ldots\}, \{\nu, \ldots\}\rangle \equiv \langle VC_{KV},\ VC_{KI}\rangle$

Combined type-level variable collections: $VC_T \in \mathcal{TVC}$

$VC_T ::= \langle VC_K, \langle\{\alpha, \ldots\}, \{\nu, \ldots\}\rangle\rangle \equiv \langle VC_K, \langle VC_{TV}, VC_{TI}\rangle\rangle$

Combined value-level variable collections: $VC \in \mathcal{VC}$

$VC ::= \langle VC_T, \{x, \ldots\}\rangle \equiv \langle VC_T,\ VC_V\rangle$

Empty collections: $\varnothing_K \in \mathcal{KVC};\ \varnothing_T \in \mathcal{TVC};\ \varnothing \in \mathcal{VC}$

$\varnothing_K \equiv \langle\{\}, \{\}\rangle$

$\varnothing_T \equiv \langle\varnothing_K, \langle\{\}, \{\}\rangle\rangle$

$\varnothing \equiv \langle\varnothing_T, \{\}\rangle$

B. Auxiliary operations

Free variables of an index: $FV \in \mathcal{I}ndex\rightarrow\wp\ \mathcal{I}Var$

$FV(n) = \{\}$

$FV(\nu) = \{\nu\}$

Free variables of a data kind: $FV \in \mathcal{DK}ind\rightarrow\mathcal{KVC}$

$FV(1) = \varnothing_K$

$FV(D) = \varnothing_K$

$FV(T) = \varnothing_K$

$FV(V) = \langle\{V\}, \{\}\rangle$

$FV(\mathbf{V}_i) = \langle\{\}, FV(i)\rangle$

$FV(\delta\times\eta) = FV(\delta) + FV(\eta)$

$FV(\kappa\rightarrow\delta) = FV(\kappa) + FV(\delta)$

Free variable of a type functional: $FV \in \mathcal{TF}nal\rightarrow\mathcal{TVC}$

$FV(\chi) = \varnothing_T$

$FV(\alpha) = \langle \varnothing_K, \langle\{\alpha\}, \{\}\rangle\rangle$

$FV(\mathbf{S}_i) = \langle\langle\{\}, FV(i)\rangle, \langle\{\}, \{\}\rangle\rangle$

$FV(\mathbf{T}_{ij}) = \langle\langle\{\}, FV(i)\rangle, \langle\{\}, FV(j)\rangle\rangle$

$FV(\theta,\omega) = FV(\theta) \cup FV(\omega)$

$FV(\theta\ \omega) = FV(\theta) \cup FV(\omega)$

$FV(\lambda\pi.\theta) = FV(\theta) \cup FV(\pi)_K \setminus FV(\pi)_{TV}$

$FV(\forall\pi.\theta) = FV(\theta) \cup FV(\pi)_K \setminus FV(\pi)_{TV}$

$FV(\forall V.\theta) = FV(\theta) \setminus \langle\langle\{V\}, \{\}\rangle, \langle\{\}, \{\}\rangle\rangle$

Free variables of a type pattern: $FV \in \mathcal{TP}atn \to \mathcal{TVC}$

$FV(\rho) = FV(\mathcal{E}(\rho))$

Free variables of an expression: $FV \in \mathcal{E}xpr \to \mathcal{VC}$

$FV(c) = \varnothing$

$FV(x) = \langle \varnothing_T, \{x\}\rangle$

$FV(\mathbf{K}_{ij}) = \langle\langle\{\}, \{i, j\}, \{\}\rangle, \{\}\rangle$

$FV(e,d) = FV(e) \cup FV(d)$

$FV(p_1 \Rightarrow e_1 \mid \ldots \mid p_n \Rightarrow e_n) = \bigcup_{1 \leqslant i \leqslant n} FV(e_i) \setminus FV(p_i)$

$FV(f\ e) = FV(f) \cup FV(e)$

$FV(\rho_1 \twoheadrightarrow e_1 \mid \ldots \mid \rho_n \twoheadrightarrow e_n) = \bigcup_{1 \leqslant i \leqslant n} FV(e_i) \setminus FV(\rho_i)$

$FV(f{\downarrow}\theta) = FV(f) \cup \langle\langle \varnothing_K, FV(\theta)\rangle, \{\}\rangle$

$FV(\mu x{:}\theta.e) = FV(e) \cup \langle\langle \varnothing_K, FV(\theta)\rangle, \{\}\rangle \setminus \langle \varnothing_T, \{x\}\rangle$

$FV(\Lambda V.e) = FV(e) \setminus \langle\langle\langle\{V\}, \{\}\rangle, \langle\{\}, \{\}\rangle\rangle, \{\}\rangle$

$FV(e{\downarrow}\kappa) = FV(f) \cup \langle\langle FV(\kappa), \langle\{\}, \{\}\rangle\rangle, \{\}\rangle$

Free variables of a pattern: $FV \in \mathcal{P}atn \to \mathcal{VC}$

$FV(p) = FV(\mathcal{E}(p))$

Free variables of an assumption: $FV \in \mathcal{A}ssm \to \mathcal{VC}$

$FV(\langle A_T, A_V\rangle) = \bigcup_{\delta \in rng(A_T)} FV(\delta) \cup \bigcup_{\theta \in rng(A_V)} FV(\theta)$

VIII. Effective Treatment of Patterns

In section IV above we gave rules for the formation of patterns which were essentially non-effective (e.g., references to "all expressions e of type τ"). Since compilers and type-checkers in a real implementation must be based on finite methods, we present in this section an effective treatment of patterns. The approach is based on the observation that in all cases above where we make reference to an infinite set of patterns, expressions, or types, we can soundly consider just a finite approximation whose elements are bounded in size by the depth of other terms appearing in the context of the reference. For pattern overlap, we give an effective operations such that $p \approx q$ iff $O(p,q)$ (and similarly for type patterns).

A. Term sorts and syntax

See syntax for patterns in section IV and for type patterns in section V above.

B. Auxiliary operations

Value pattern depth: $\mathcal{D} \in \mathcal{P}atn \to \mathcal{N}at$

$\mathcal{D}(c) = 0$

$\mathcal{D}(x{:}\tau) = 0$

$\mathcal{D}(\mathbf{K}_{ij}{\downarrow}\theta\ p) = 1 + \mathcal{D}(p)$

$\mathcal{D}(p,q) = 1 + \max(\mathcal{D}(p), \mathcal{D}(q))$

Generated patterns: $\mathcal{G}^n \in \mathcal{T}ype \to \wp(\mathcal{P}atn)$

{NB: this definition assumes that the type argument has been reduced to (head-) normal form}

$\mathcal{G}^0(\tau) = \{x{:}\tau\}$ (fresh x)

$\mathcal{G}^{n+1}(1) = \{*\}$

$\mathcal{G}^{n+1}(\tau \times \sigma) = \{p,q \mid p \in \mathcal{G}^n(\tau),\ q \in \mathcal{G}^n(\sigma)\}$

$\mathcal{G}^{n+1}(\tau \to \sigma) = \{x{:}\tau \to \sigma\}$

$\mathcal{G}^{n+1}(\alpha) = \{x{:}\alpha\}$ (fresh x)

$\mathcal{G}^{n+1}(\alpha\ \theta) = \{x{:}\alpha\ \theta\}$ (fresh x)

$\mathcal{G}^{n+1}(\mathbf{S}_n\ \theta) = \{\mathbf{K}_{nm}{\downarrow}\theta\ p \mid 1 \leqslant m \leqslant |C_{VK}(n)|,\ \mathbf{T}_{nm}\ \theta \twoheadrightarrow \omega,\ p \in \mathcal{G}^n(\omega)\}$

$\mathcal{G}^{n+1}(\mathbf{S}_v\ \theta) = \{\mathbf{K}_{lm}{\downarrow}\theta\ p \mid 1 \leqslant l \leqslant |C_{TS}|,\ 1 \leqslant m \leqslant |C_{VK}(l)|,\ \mathbf{T}_{lm}\ \theta \twoheadrightarrow \omega,\ p \in \mathcal{G}^n(\omega)\}$

$\mathcal{G}^{n+1}(\mathbf{T}_{ij}\ \theta) = \{x{:}\mathbf{T}_{ij}\ \theta\}$

$\mathcal{G}^{n+1}(\forall\pi.\theta) = \{x{:}\forall\pi.\theta\}$ (fresh x)

$\mathcal{G}^{n+1}(\forall V.\theta) = \{x{:}\forall V.\theta\}$ (fresh x)

Value pattern coverage: $\{p,\ldots\} \succ \tau$

$\{p_1, \ldots, p_m\} \succ \tau$ iff

for all $q \in \mathcal{G}^n(\tau)$, some $p_i \sqsubseteq q$, where $n = \max\{\mathcal{D}(p_i) \mid 1 \leqslant i \leqslant m\}$

Type pattern depth: $\mathcal{D} \in \mathcal{TPatn} \to \mathcal{N}at$

$\mathcal{D}(\Phi\ \rho_1 \cdots \rho_n) = 1+\max(\mathcal{D}(\rho_1), \ldots, \mathcal{D}(\rho_n))$

$\mathcal{D}(\rho,\upsilon) = 1+\max(\mathcal{D}(\rho),\mathcal{D}(\upsilon))$

$\mathcal{D}(\lambda\pi.\rho) = \mathcal{D}(\rho)$

$\mathcal{D}(\psi\ \rho_1 \cdots \rho_n : \delta) = 0$

Fresh type parameters: $\mathcal{P} \in \mathcal{K}ind \to \mathcal{TP}arm$

$\mathcal{P}(1) = *$

$\mathcal{P}(T) = \alpha{:}T$ (fresh α)

$\mathcal{P}(V) = \alpha{:}V$ (fresh α)

$\mathcal{P}(\mathbf{V}_i) = \alpha{:}\mathbf{V}_i$ (fresh α)

$\mathcal{P}(\kappa\times\eta) = \mathcal{P}(\kappa)\ ,\ \mathcal{P}(\eta)$

$\mathcal{P}(\kappa\to\eta) = \alpha : \kappa\to\eta$ (fresh α)

Generated type patterns: $\mathcal{G}_\Pi^n \in \mathcal{D}\mathcal{K}ind \to \wp(\mathcal{TP}atn)$

$\mathcal{G}_\Pi^0(\delta) = \{\psi\ \rho_1 \cdots \rho_n : \delta\}$, where $\langle\rho_1, \ldots, \rho_n\rangle = \mathcal{E}(\Pi)$ (fresh ψ)

{NB: arrow kinds are handled by another clause below}

$\mathcal{G}_\Pi^{n+1}(1) = \{*\}$

$\mathcal{G}_\Pi^{n+1}(D) = \{1\} \cup \{(\rho_1\times\rho_2) \mid \rho_i \in \mathcal{G}_\Pi^n(D)\} \cup \mathcal{S}_\Pi^{n+1}$

$\quad \cup \bigcup \{\mathcal{V}_\Pi^{n+1}(\{\alpha\},\kappa) \mid \Pi(\alpha)=\kappa\}$

$\mathcal{G}_\Pi^{n+1}(T) = \{\alpha{:}T\}$ (fresh α)

$\mathcal{G}_\Pi^{n+1}(V) = \{\alpha{:}V\}$ (fresh α)

$\mathcal{G}_\Pi^{n+1}(\mathbf{V}_i) = \{\alpha{:}\mathbf{V}_i\}$ (fresh α)

$\mathcal{G}_\Pi^{n+1}(\delta\times\eta) = \{(\rho,\upsilon) \mid \rho \in \mathcal{G}_\Pi^n(\delta) + \upsilon \in \mathcal{G}_\Pi^n(\eta)\}$

$\mathcal{G}_\Pi^n(\kappa\to\delta) = \{\lambda\pi.\rho \mid \pi = \mathcal{P}(\kappa),\ \rho \in \mathcal{G}_{\Pi,\pi}^n(\delta)\}$

Structure patterns: $\mathcal{S}_\Pi^{n+1} \in \wp(\mathcal{TP}atn)$

$\mathcal{S}_\Pi^{n+1} = \{\mathbf{S}_m\ \ \rho \mid 1 \leqslant m \leqslant |C_{TS}|,\ C_{TS}(m) = \kappa \to D,\ \rho \in \mathcal{G}_\Pi^n(\kappa^-)\}$

Variable patterns: $\mathcal{V}_\Pi^n \in \wp(\mathcal{TP}atn)\times\mathcal{K}ind \to \wp(\mathcal{TP}atn)$

$\mathcal{V}_\Pi^{n+1}(R,T) = R$

$\mathcal{V}_\Pi^{n+1}(R,\kappa\to\eta) = \mathcal{V}_\Pi^{n+1}(\{\rho\ \upsilon \mid \rho \in R,\ \upsilon \in \mathcal{G}_\Pi^n(\kappa^-)\},\eta)$

Coverage: $\{\rho,\ldots\} \succ \delta$

$\{\rho_1, \ldots, \rho_m\} \succ \delta$ iff for all $\upsilon \in \mathcal{G}_{\langle\rangle}^n(\delta)$, some $\rho_i \sqsubseteq \upsilon$, where $n = \max\{\mathcal{D}(\rho_i) \mid 1 \leqslant i \leqslant m\}$

Index overlap: $\mathcal{O} \in \mathcal{I}ndex\times\mathcal{I}ndex \to \mathcal{B}ool$

$\mathcal{O}(\mu,j) = \text{true}$

$\mathcal{O}(i,\nu) = \text{true}$

$\mathcal{O}(n,n) = \text{true}$

$\mathcal{O}(i,j) = \text{false, otherwise}$

Pattern overlap: $\mathcal{O} \in \mathcal{P}atn\times\mathcal{P}atn \to \mathcal{B}ool$

$\mathcal{O}(c,c) = \text{true}$

$\mathcal{O}(x{:}\theta,q) = \text{true}$

$\mathcal{O}(p,y{:}\omega) = \text{true}$

$\mathcal{O}(\mathbf{K}_{ij}{\downarrow}\theta\ p,\ \mathbf{K}_{hk}{\downarrow}\omega\ q) = \mathcal{O}(i,h)$ and $\mathcal{O}(j,k)$ and $\mathcal{O}(p,q)$

$\mathcal{O}((p,p'),(q,q')) = \mathcal{O}(p,q)$ and $\mathcal{O}(p',q')$

$\mathcal{O}(p,q) = \text{false, otherwise}$

Type pattern overlap: $\mathcal{O} \in \mathcal{TP}atn\times\mathcal{TP}atn \to \mathcal{B}ool$

$\mathcal{O}(\chi,\chi) = \text{true}$

$\mathcal{O}(\alpha,\alpha) = \text{true}$ *{NB: same α}*

$\mathcal{O}(\mathbf{S}_i, \mathbf{S}_j) = \mathcal{O}(i,j)$

$\mathcal{O}(\Phi\ \rho_1 \cdots \rho_n, \Theta\ \upsilon_1 \cdots \upsilon_n) = \mathcal{O}(\Phi,\Theta)$ and $\mathcal{O}(\rho_1,\upsilon_1)$ … and $\mathcal{O}(\rho_n,\upsilon_n)$

$\mathcal{O}((\rho,\rho'),(\upsilon,\upsilon')) = \mathcal{O}(\rho,\upsilon)$ and $\mathcal{O}(\rho',\upsilon')$

$\mathcal{O}(\lambda\pi.\rho,\lambda\pi'.\upsilon) = \mathcal{O}(\rho,\upsilon[\pi/\pi'])$

$\mathcal{O}(\rho, (\zeta\ \upsilon_1 \cdots \upsilon_n){:}\gamma) = \text{true}$

$\mathcal{O}((\psi\ \rho_1 \cdots \rho_n){:}\delta, \upsilon) = \text{true}$

$\mathcal{O}(\rho,\upsilon) = \text{false, otherwise}$

Appendix C: Definitions for the Structurally Polymorphic Calculus

The first-order pattern calculus with structural polymorphism is based on a the higher-order, explicit calculus, but with a few changes: kinds are restricted to first-order (i.e., no nested function spaces); we include now the "intersection" types to represent structural polymorphism; we use *type schemes* to represent the usual parametric polymorphism; and there are two sorts of expressions, *raw* and *decorated.* The typing judgements for expressions specify a transformation from raw to decorated expressions; only the latter are considered meaningful in the context of reduction and conversion.
The following diagram illustrates the relations between the various sorts of the structurally polymorphic calculus. Note especially that assumptions and settings associate type *schemes* with variables and constants; that raw and decorated expressions participate jointly in their formation relation through types; and that type schemes are formed relative to the formation of their type body, which must have kind T (we do not show here the details of the usual data type order sorts).

The presentation in this appendix follows the approach of Chapter 8, with minimal explicit type constraints required for the spoly-definition form of raw expressions. The system with fully implicit terms differes only in minor respects—see Chapter 7 for details. Since much of the presentation of this calculus is the same as the one for the explicitly-typed calculus of Appendix B, we concentrate here on the changes. The overall structure of the presentation follows this outline:

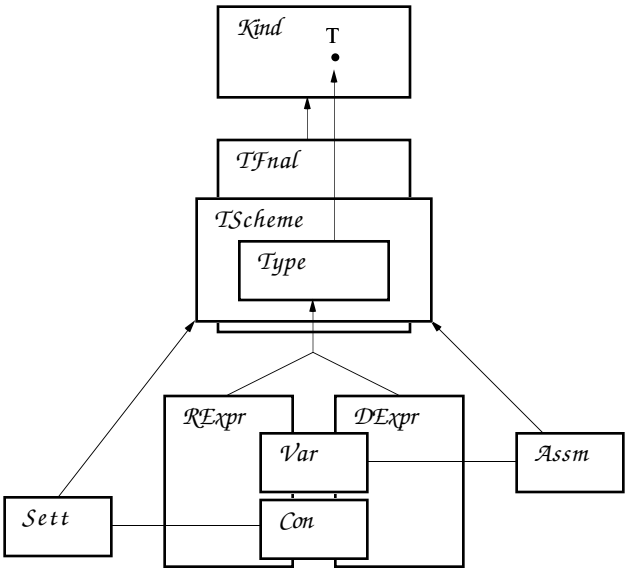

Figure 1: Formation relations for the first-order calculus

I. Kinds
II. Types, Type Functionals and Type Schemes
III. Value Expressions
IV. Value Patterns
V. Keyword and Type Patterns
VI. Matching and Conversion
VII. Free Variables
VIII. Effective Treatment of Patterns
IX. Type Inference Algorithm

I. Kinds

As usual, kinds are used to control the formation of types and type functionals; since this is a first-order calculus, however, we use kinds primarily to restrict type functionals to just operating on types, tuples of types, etc. Since it makes the presentation more like that of Appendix B, we use full products over kinds here, although we actually account only for n-tuples of type parameters in the treatment of keywords below (a suitable transformation between the two is easily defined, flattening the product "trees" into lists in one direction and yieldingthe usual canonical representation of lists as trees in the the other).

A. Term sorts and syntax *{as in Appendix B}*

B. Auxiliary operations

Order of data kinds: $O \in \mathcal{DKind} \to \mathcal{Nat}$

$$O(1) = 0 \qquad O(\mathrm{B}) = 0$$

$$O(\mathrm{V}) = 0 \qquad O(\mathbf{V}_i) = 0$$

$$O(\delta \times \eta) = \max(O(\delta), O(\eta))$$

$$O(\kappa \to \delta) = \max(1 + O(\kappa), O(\delta))$$

C. Inference rules *{as in Appendix B, except where given}*

Data kind formation: $\mathrm{A_K} \vdash \delta$ dkind

$$[dkind]\ \frac{O(\delta) \leqslant 1 \qquad \mathrm{FV_{KV}}(\delta) \subseteq \mathrm{A_{KV}} \qquad \mathrm{FV_{KI}}(\delta) \subseteq \mathrm{A_{KI}}}{\mathrm{A_K} \vdash \delta \text{ dkind}}$$

II. Types, Type Functionals and Type Schemes

The syntax of the type level includes also now type schemes, which are used to control parametric polymorphism, and "intersection types", which are used to control structural polymorphism (which is justified by recursive case analysis). Note that the intersection types may appear freely in types, unlike with type schemes, which force universal quantifications to appear only outermost.

A. Term sorts and syntax *{as in Appendix B, except where given}*

Type functionals: $\theta, \omega \in \mathcal{TFnal}$

$$\theta ::= \chi \mid \alpha \mid \mathbf{S}_i \mid \mathbf{T}_{ij} \mid \theta,\theta \mid \theta\ \theta \mid \lambda\pi.\theta \mid \bigcap\alpha{:}\delta.\theta \mid \bigcap\mathbf{S}_v.\theta$$

Type schemes: $\sigma \in \mathcal{TScheme}$

$$\sigma ::= \theta \mid \forall\alpha{:}\mathrm{T},\ldots,\alpha{:}\mathrm{T}.\ \theta$$

B. Auxiliary operations *{as in Appendix B, except where given}*

Instantiation: $\sigma \sqsubseteq \sigma'$

$\sigma \sqsubseteq \sigma'$ iff $\sigma[S] = \sigma'$

Generic instantiation: $\sigma \preccurlyeq \sigma'$

$\forall \alpha_1,\ldots,\alpha_n.\, \tau \preccurlyeq \forall \beta_1,\ldots,\beta_m.\, \tau'$

iff $\tau[S] = \tau'$ for $dom(S) \subseteq \{\alpha_1,\ldots,\alpha_n\}$ and $\{\beta_1,\ldots,\beta_m\} \cap FV(\sigma)$

C. Inference rules *{as in Appendix B, except where given; the changes to context associations just restrict to first-order kinds}*

Constant setting formation: $\vdash C_{TC}$ tcsett

$$[tcsett]\quad \frac{\vdash \delta_1 \text{ dkind} \quad \cdots \quad \vdash \delta_n \text{ dkind} \qquad O(\delta_1)\leqslant 1 \quad \cdots \quad O(\delta_n)\leqslant 1}{\vdash \{\chi_1{:}\delta_1, \ldots, \chi_n{:}\delta_n\} \text{ tcsett}}$$

Constructor setting formation: $\vdash C_{TS}$ tssett

$$[tssett]\quad \frac{\vdash \kappa_1 \text{ dkind} \quad \cdots \quad \vdash \kappa_n \text{ dkind} \qquad O(\kappa_1)=0 \quad \cdots \quad O(\kappa_n)=0}{\vdash \langle \kappa_1 \to B_1, \ldots, \kappa_n \to B_n \rangle \text{ tssett}}$$

Type assumption formation: $\vdash A_T$ tassm

$$[tassm]\quad \frac{\vdash \delta_1 \text{ dkind} \quad \cdots \quad \vdash \delta_n \text{ dkind} \qquad O(\delta_1)\leqslant 1 \quad \cdots \quad O(\delta_n)\leqslant 1}{\vdash \{\alpha_1{:}\delta_1, \ldots, \alpha_n{:}\delta_n\} \text{ tassm}}$$

Type formation: $A_T \vdash \theta : \kappa$

$$[t\text{-}lam]\quad \frac{A_K \vdash \pi : \kappa \qquad A +_{TV} \mathcal{A}(\pi)^- \vdash \theta : \delta}{A \vdash (\lambda\pi.\theta) : \kappa \to \delta}$$

$$[un\text{-}tv]\quad \frac{\delta = \delta^- \qquad O(\delta)=0 \qquad A +_{TV} \{\alpha{:}\delta\} \vdash \theta : T}{A \vdash (\bigcap \alpha{:}\delta.\theta) : T}$$

$$[un\text{-}sv]\quad \frac{A \vdash \theta : T}{A \vdash (\bigcap \mathbf{S}_v.\theta) : T}$$

Type scheme formation: $A_T \vdash \sigma$ scheme

$$[t\text{-}lam]\quad \frac{A +_{TV} \{\alpha_1{:}T, \ldots, \alpha_n{:}T\} \vdash \theta : T}{A \vdash (\forall \alpha_1{:}T, \ldots, \alpha_n{:}T.\theta) \text{ scheme}}$$

III. Value Expressions

In the structurally polymorphic calculus we consider two sorts of expressions, raw and decorated. The former are "transformed" into the latter during type-checking, and only decorated expressions are considered for reduction. Less operationally, we use a well-formation judgement which relates raw and decorated terms with a type. Raw expressions include the spoly-definition form, which is transformed into a recursive type-pattern abstraction as a decorated term (note that we describe here the system with minimal explicit constraints). The let-definition form is the familiar one used to express implicit parametric polymorphism. Type assumptions are changed to associate type schemes to free value variables.

A. Term sorts and syntax

Combined value-level assumptions: $A \in \mathcal{A}ssm$

$A ::= \langle A_T, \{x{:}\sigma, \ldots\}\rangle \equiv \langle A_T, A_V\rangle$

Raw value expressions: $e, d \in \mathcal{E}xpr$

$e ::= c \mid x \mid \mathbf{K}_{ij} \mid e,e \mid e\ e \mid (p \Rightarrow e)^+ \mid \mu x.e \mid$ **spoly** x **as** $(\text{key } e)^+$ $\mid$ **let** x = e **in** e

Decorated value expressions: $e, d \in \mathcal{E}xpr$

$d ::= c \mid x \mid \mathbf{K}_{ij} \mid d,d \mid d\ d \mid (p \Rightarrow d)^+ \mid \mu x.e \mid d{\downarrow}\theta \mid (\rho \twoheadrightarrow d)^+ \mid$ **let** x = d **in** d

B. Auxiliary operations

Explicit instantiation: $inst(\sigma)$

$inst(\forall \alpha_1,\ldots,\alpha_n.\tau) = \tau[\beta_1/\alpha_1,\ldots,\beta_n/\alpha_n]$, fresh β_i

Generalization: $gen(\tau, A_V)$

$gen(\tau, A_V) =$

$\forall \alpha_1{:}T,\ldots,\alpha_n{:}T.\bigcap \mathbf{S}_{v_1}{:}\kappa_1,\ldots,\mathbf{S}_{v_m}{:}\kappa_m.$

$\bigcap \beta_1{:}D,\ldots,\beta_k{:}D.\tau$

where

$\{\alpha_1,\ldots,\alpha_n\} = \{\alpha \in FV_{TV}(\tau) \setminus FV_{TV}(A_V) \mid A_{TV}(\alpha) = T\}$,

$\{v_1,\ldots,v_m\} = FV_{TI}(\tau) \setminus FV_{TI}(A_V)$,

$\kappa_i = A_{TS}(\mathbf{S}_{v_i})$, for $1 \leqslant i \leqslant m$, and

$\{\beta_1,\ldots,\beta_k\} = \{\beta \in FV_{TV}(\tau) \setminus FV_{TV}(A_V) \mid A_{TV}(\beta) = D\}$

C. Inference rules

Value constructor setting formation: $\vdash C_{VK}$ vksett

$$[vks]\quad \frac{\begin{array}{c} n = |C_{TS}| \\ m_1 \geqslant 1 \quad \cdots \quad m_n \geqslant 1 \\ \{\} \vdash \theta_{11} : \delta_{11} \qquad \delta_{11} \leqslant C_{TS}(1) \qquad O(\delta_{11}) \leqslant 1 \\ \cdots \qquad\qquad \cdots \qquad\qquad \cdots \\ \{\} \vdash \theta_{nm_n} : \delta_{nm_n} \qquad \delta_{nm_n} \leqslant C_{TS}(n) \qquad O(\delta_{nm_n}) \leqslant 1 \end{array}}{\vdash \langle\langle \theta_{11}, \ldots, \theta_{1m_1}\rangle, \ldots, \langle \theta_{n1}, \ldots, \theta_{nm_n}\rangle\rangle \text{ vksett}}$$

Value assumption formation: $\vdash A\ \text{assm}$

$$\vdash A_T\ \text{tassm}$$

$$[assm]\ \frac{A_T \vdash \sigma_1\ \text{scheme} \quad \cdots \quad A_T \vdash \sigma_n\ \text{scheme}}{\vdash \langle A_T, \{x_1{:}\sigma_n, \ldots, x_n{:}\sigma_n\}\rangle\ \text{assm}}$$

Expression formation abbreviation: $A \vdash e : \tau$

$A \vdash e : \tau \triangleq A \vdash e \mapsto e : \tau$

where the right-most occurrence of d is a (degenerately) decorated expression

Expression formation: $A \vdash e \mapsto d : \tau$

$$[con]\quad A \vdash c : \text{inst}(C_{VC}(c))$$

$$[var]\quad A \vdash x : \text{inst}(A_V(x))$$

$$[cc\text{-}con]\ \frac{\alpha \notin FV(A) \qquad C_{TS}(n) = \kappa \to B \qquad (C_{VK}(n)(m))\ \alpha = \theta}{A \vdash \mathbf{K}_{nm} : \theta \to \mathbf{S}_n\ \alpha}$$

$$[cv\text{-}con]\ \frac{C_{TS}(n) = \kappa \to B \qquad \alpha \notin FV(A)}{A \vdash \mathbf{K}_{n\mu} : \mathbf{T}_{n\mu}\ \alpha \to \mathbf{S}_n\ \alpha}$$

$$[vv\text{-}con]\ \frac{\alpha \notin FV(A)}{A \vdash \mathbf{K}_{v\mu} : \mathbf{V}_v.\ \mathbf{T}_{v\mu}\ \alpha \to \mathbf{S}_v\ \alpha}$$

$$[pair]\ \frac{A \vdash e \mapsto d : \tau \qquad A \vdash e' \mapsto d' : \sigma}{A \vdash (e,e') \mapsto (d,d') : \tau \times \sigma}$$

$$[abs]\ \frac{\{p_1, \ldots, p_n\}\ \text{compatible} \quad \{p_1, \ldots, p_n\} \succ \tau \quad A' \vdash p_i : \tau \quad (1 \leqslant i \leqslant n) \quad A + A' \vdash e_i \mapsto d_i : \sigma \quad (1 \leqslant i \leqslant n)}{A \vdash (\cdots \mid p_i \Rightarrow e_i \mid \cdots) \mapsto (\cdots \mid p_i \Rightarrow d_i \mid \cdots) : \tau \to \sigma}$$

$$[app]\ \frac{A \vdash f \mapsto g : \sigma \to \tau \qquad A \vdash e \mapsto d : \sigma}{A \vdash (f\ e) \mapsto (g\ d) : \tau}$$

$$[rec]\ \frac{A +_V \{x{:}\tau\} \vdash e \mapsto d : \tau}{A \vdash (\mu x.e) \mapsto (\mu x.d) : \tau}$$

$$[let]\ \frac{A \vdash e \mapsto d : \tau \qquad A + \{x{:}\text{gen}(\tau, A)\} \vdash e' \mapsto d' : \tau'}{A \vdash (\textbf{let}\ x = e\ \textbf{in}\ e') \mapsto (\textbf{let}\ x = d\ \textbf{in}\ d') : \tau'}$$

$$[tv\text{-}inst]\ \frac{A \vdash f \mapsto g : \bigcap \alpha{:}\delta.\tau \qquad A_T \vdash \theta : \delta}{A \vdash f \mapsto (g{\downarrow}\theta) : \tau\ [\theta/\alpha]}$$

$$[tv\text{-}inst]\ \frac{A \vdash f \mapsto g : \bigcap \mathbf{S}_v.\tau \qquad A_T \vdash \theta : \mathbf{V}_i}{A \vdash f \mapsto (g{\downarrow}\theta) : \tau\ [i/v]}$$

$$[spoly1]\ \frac{\{key_1, \ldots, key_n\}\ \text{compatible} \quad \{key_1, \ldots, key_n\} \succ \delta \quad \left(\{\alpha\} \cup \bigcup_i FV(\rho_i)_T\right) \ncong FV(A)_T \quad \mathcal{TP}(D, key_i) = \rho_i \quad (1 \leqslant i \leqslant n) \quad A + \mathcal{A}(\rho_i) + \{x{:}\bigcap \alpha{:}D.\tau\} \vdash e_i \mapsto d_i : \tau[\rho_i/\alpha] \quad (1 \leqslant i \leqslant n)}{A \vdash (\textbf{spoly}\ x{:}\bigcap \alpha{:}D.\tau\ \textbf{as}\ (\cdots \mid key_i\ e_i \mid \cdots)) \mapsto (\mu x.\cdots \mid \rho_i \Rightarrow\!\!\!\!\!\!\Rightarrow d_i \mid \cdots) : \bigcap \alpha{:}D.\tau}$$

$$[spoly2]\ \frac{\{key_1, \ldots, key_n\}\ \text{compatible} \quad \{key_1, \ldots, key_n\} \succ \kappa \to D \quad \left(\{\alpha\} \cup \bigcup_i FV(\rho_i)_T\right) \ncong FV(A)_T \quad \mathcal{TP}(\kappa \to D, key_i) = \rho_i \quad (1 \leqslant i \leqslant n) \quad A + \mathcal{A}(\rho_i) + \{x{:}\bigcap \alpha{:}\kappa \to D.\tau\} \vdash e_i \mapsto d_i : \tau[\rho_i/\alpha] \quad (1 \leqslant i \leqslant n)}{A \vdash (\textbf{spoly}\ x{:}\bigcap \alpha{:}\kappa \to D.\tau\ \textbf{as}\ (\cdots \mid key_i\ e_i \mid \cdots)) \mapsto (\mu x.\cdots \mid \rho_i \Rightarrow\!\!\!\!\!\!\Rightarrow d_i \mid \cdots) : \bigcap \mathbf{S}_v{:}\kappa \to D.\tau}$$

IV. Value Patterns

The only changes we need to the syntax of value patterns are to eliminate explicit type constraints and instantiations. With this change, patterns have now become just a subset of expressions, and thus their formation is just as expression formation, so that we no longer have need for the usual assumption projection and erasure operations (note the subsequent changes in the pattern abstraction rule above).

A. Term sorts and syntax

Value patterns: $p, q \in \mathcal{P}atn \subseteq \mathcal{E}xpr$

$p ::= c \mid x \mid p\ ,\ p \mid \mathbf{K}_{ij}\ p$

B. Auxiliary operations and relations *{same as in Appendix B, modulo the explicit typing in patterns}*

C. Inference rules *{same as in Appendix B, except that we need no separate pattern-formation judgement}*

V. Type and Keyword Patterns

According to the formation rule for the spoly-definition form above, keyword patterns are used as the basis for determining whether the appropriate cases are covered completely and unambiguously. This is in keeping with our interpretation that programmers will write raw expressions and that decorated expressions will be obtained from the raw ones as a part of the

type inference process. Type patterns will be generated automatically as constituents of the resulting decorated expression, and so we need only make the appropriate checks on the programmer's keyword patterns as input. The operation defined below to associate fixed type patterns with keywords realizes the restriction of functional domain kinds to "flat" products.

A. Term sorts and syntax *{type patterns are the same as in Appendix B}*

Keyword patterns: $key \in \mathcal{KP}atn$

$key ::= \mathbf{unit} \mid \mathbf{pair} \mid \mathbf{struc} \mid \mathbf{comp\#i}$

B. Auxiliary operations and relations

Keyword pattern coverage: $\{key, \ldots\} \succ \delta$

$keys \succ T^n \to D$ iff $\{\mathbf{unit}, \mathbf{pair}, \mathbf{struc}\} \subseteq keys$ and for all $1 \leqslant i \leqslant n$, $\mathbf{comp\#i} \in keys$

{NB: here we take $D \triangleq T^0 \to D$ *for simplicity}*

Keyword pattern compatibility: $\{key, \ldots\}$ compatible

$\{key_1, \ldots, key_n\}$ compatible iff for all $1 \leqslant i \neq j \leqslant n$, $key_i \neq key_j$

Type patterns for keywords: $\mathcal{TP} \in \mathcal{KP}atn \times \mathcal{DK}ind \to \mathcal{TP}atn$

$\mathcal{TP}(D, \mathbf{unit}) = 1$

$\mathcal{TP}(D, \mathbf{pair}) = \alpha{:}D \times \beta{:}D$ (fresh α, β)

$\mathcal{TP}(D, \mathbf{unit}) = \mathbf{S}_\nu\ (\chi{:}\mathbf{V}_\nu)$ (fresh ν, χ)

$\mathcal{TP}(\kappa_1 \times \cdots \times \kappa_n \to D, \mathbf{unit}) = \lambda \varepsilon_1{:}\kappa_1, \ldots, \varepsilon_n{:}\kappa_n.\ 1$

$\mathcal{TP}(\kappa_1 \times \cdots \times \kappa_n \to D, \mathbf{pair}) =$

$\lambda \varepsilon_1{:}\kappa_1, \ldots, \varepsilon_n{:}\kappa_n.\ (\psi\ \varepsilon_1\ \ldots\ \varepsilon_n{:}D) \times (\zeta\ \varepsilon_1\ \ldots\ \varepsilon_n{:}D)$ (fresh ψ, ζ)

$\mathcal{TP}(\kappa_1 \times \cdots \times \kappa_n \to D, \mathbf{struc}) = \lambda \varepsilon_1{:}\kappa_1, \ldots, \varepsilon_n{:}\kappa_n.\ \mathbf{S}_\nu\ (\chi\ \varepsilon_1\ \ldots\ \varepsilon_n{:}\mathbf{V}_\nu)$ (fresh ν, χ)

$\mathcal{TP}(\kappa_1 \times \cdots \times \kappa_n \to D, \mathbf{comp\#i}) = \lambda \varepsilon_1{:}\kappa_1, \ldots, \varepsilon_n{:}\kappa_n.\ \varepsilon_i$

C. Inference rules *(none).*

VI. Matching and Conversion

As mentioned above, matching, reduction and conversion are defined in this calculus only for decorated expressions, which are the well-typed translations of some raw expression. Since decorated expressions include types and type patterns as constituents, we still need matching between types and type patterns, but there are no changes required for the associated operations. Note that there is no notion of reduction for keyword patterns, since they do not appear in decorated expressions.

A. Term sorts and syntax *{as in Appendix B}*

B. Auxiliary operations *{as in Appendix B, modulo the explicit typing in patterns}*

C. Inference rules *{as in Appendix B, modulo explicit typing, and excepting the let-definition construct. Note that we need not define conversion for spoly definitions, since these are a form of raw term}*

Expression conversion: $e = e'$

[let] $\mathbf{let}\ x = d\ \mathbf{in}\ d' = d'\ [d/x]$

VII. Free Variables

We need only to account for the new definitional forms of expressions and the new intersection types in describing free variable operations.

A. Term sorts and syntax *{as in Appendix B, modulo the explicit typing in patterns}*

B. Auxiliary operations *{as in Appendix B, but with these additions}*

Free variable of a type functional: $FV \in \mathcal{TF}nal \to \mathcal{TVC}$

$FV(\bigcap \alpha{:}\delta.\theta) = FV(\theta) \cup FV(\delta)_K \setminus \{\alpha\}$

$FV(\bigcap \mathbf{S}_\nu.\theta) = FV(\theta) \setminus \langle\langle\{\}, \{\nu\}\rangle, \langle\{\}, \{\}\rangle\rangle$

Free variable of a type scheme: $FV \in \mathcal{TS}cheme \to \mathcal{TVC}$

$FV(\forall \alpha_1{:}T, \ldots, \alpha_n{:}T.\theta) = FV(\theta) \setminus \{\alpha_1, \ldots, \alpha_n\}$

Free variables of an expression: $FV \in \mathcal{E}xpr \to \mathcal{VC}$

$FV(\mathbf{let}\ x = d\ \mathbf{in}\ d') = FV(d') \cup (FV(d) \setminus \{x\})$

$FV(\mathbf{spoly}\ x\ \mathbf{as}\ (key\ e)^+) = FV(d') \cup (FV(d) \setminus \{x\})$

Free variables of an assumption: $FV \in \mathcal{A}ssm \to \mathcal{VC}$

$FV(\langle A_T, A_V \rangle) = \bigcup_{\delta \in rng(A_T)} FV(\delta) \cup \bigcup_{\theta \in rng(A_V)} FV(\theta)$

VIII. Effective Treatment of Patterns

We need only minor changes to acheive an effective treatment of patterns in this calculus: in particular, our account of keywords above is already effective, so that we need only extend the definition of the pattern-set generation operation to intersection types. Note that the resultant patterns have no explicit constraints in this calculus.

A. Term sorts and syntax *{as in Appendix B}*

B. Auxiliary operations *{as in Appendix B, modulo explicit type constraints, but with these additions}*

Generated patterns: $\mathcal{G}^n \in \mathcal{T}ype \to \wp(\mathcal{P}atn)$

$\mathcal{G}^{n+1}(\bigcap \alpha{:}\delta.\theta) = \{x\}$ (fresh x)

$\mathcal{G}^{n+1}(\bigcap \mathbf{S}_\nu{:}\kappa \to D.\theta) = \{x\}$ (fresh x)

IX. Type Inference Algorithm

In this section we present the type inference algorithm which transforms raw expressions into decorated ones, while determining that they are (jointly) well-typed. As in the presentation in Chapters 7 and 8, we change our notation for substitution here to place the argument term on the right, following traditional usage in this area. Note also the we use a notion of unification which is extended to operate on second-order terms.

A. Term sorts and syntax *{expressions as above, type substitutions as in Appendix B}*

B. Auxiliary operations

Decoration: $\mathcal{D} \in \mathcal{DE}xpr \times Type \times Assm \to \mathcal{DE}xpr \times Type \times Assm$

$\mathcal{D}(d, \tau, A) =$
  if $\tau = \bigcap \alpha{:}\delta.\tau'$ then $\mathcal{D}(d{\downarrow}\beta, \tau'[\beta/\alpha], A)$, fresh β
    else (d, τ, A)

Pattern assumptions: $\mathcal{V} \in Patn \to Patn \times Type \times Assm$

$\mathcal{V}(c) = (c, inst(C_{VC}(c)), \{\})$;

$\mathcal{V}(x) = (x, \alpha, \{x{:}\alpha\})$;

$\mathcal{V}(\mathbf{K}_{nm}\ p) =$
  let $(p', \tau, A) = \mathcal{V}(p)$
  let $\theta = nf(C_{VK}(n)(m)\ \alpha)$, fresh α
  let $\upsilon = \theta \to \mathbf{S}_n\ \alpha$
  let $U = unify(\upsilon, \tau \to \beta)$, fresh β
  return $(\mathbf{K}_{nm}\ p', U\beta, UA)$;

$\mathcal{V}(\mathbf{K}_{i\mu}\ p) =$
  let $(p', \tau, A) = \mathcal{V}(p)$
  let $U = unify(\mathbf{K}_{i\mu}\ \alpha \to \mathbf{S}_i\ \alpha, \tau \to \beta)$, fresh α, β
  return $(U(\mathbf{K}_{i\mu})\ p', U\beta, UA)$;

$\mathcal{V}(p,q) =$
  let $(p', \tau, A) = \mathcal{V}(p)$
  let $(q', \upsilon, B) = \mathcal{V}(q)$
  return $((p', q'), (\tau \times \upsilon), A+B)$;

Type inference: $\mathcal{W} \in \mathcal{RE}xpr \times Assm \to \mathcal{DE}xpr \times Type \times TSubn$

$\mathcal{W}(c, A) = (c, inst(C_{VC}(c)), [])$;

$\mathcal{W}(x, A) = \mathcal{D}(x, inst(A(x)), [])$;

$\mathcal{W}(\mathbf{K}_{nm}, A) =$
  let $\theta = nf(C_{VK}(n)(m)\ \alpha)$, fresh α
  return $(\mathbf{K}_{ij}, \theta \to Sn\ \alpha, [])$;

$\mathcal{W}(\mathbf{K}_{i\mu}, A) = (\mathbf{K}_{ij}, \mathbf{K}_{i\mu}\ \alpha \to \mathbf{S}_i\ \alpha, [])$, fresh α ;

$\mathcal{W}((e, e'), A) =$
  let $(d, \rho, S) = \mathcal{W}(e, A)$
  let $(d', \tau, T) = \mathcal{W}(e', SA)$
  return $\mathcal{D}((Td, d'), T\rho \times \tau, TS)$;

$\mathcal{W}(f\ e, A) =$
  let $(g, \rho, S) = \mathcal{W}(f, A)$
  let $(d, \tau, T) = \mathcal{W}(e, SA)$
  let $U = unify(T\rho, \tau \to \beta)$, fresh β
  return $\mathcal{D}(UTg\ Ud, U\beta, UTS)$;

$\mathcal{W}(\mu x.e, A) =$
  let $(d, \tau, S) = \mathcal{W}(e, A+\{x{:}\alpha\})$, fresh α
  let $U = unify(S\alpha, \tau)$
  return $\mathcal{D}(Ud, U\tau, US)$;

$\mathcal{W}(\textbf{let}\ x{=}e\ \textbf{in}\ e', A) =$
  let $(d, \rho, S) = \mathcal{W}(e, A)$
  let $(d', \tau, T) = \mathcal{W}(e', SA+\{x{:}gen(\rho, SA)\})$
  return $\mathcal{D}(\textbf{let}\ x{=}Td\ \textbf{in}\ d', \tau, TS)$;

$\mathcal{W}((p_1 \Rightarrow e_1 \mid \ldots \mid p_n \Rightarrow e_n), A) =$
  $g_0 = \cdot$; $\sigma_0 = \beta$ (fresh β) ; $R_0 = Id$; $A_0 = A$
  for $i = 1$ to n:
    let $A_i = R_{i-1}A_{i-1}$
      $(q_i, \rho_i, B_i) = \mathcal{V}(p_i)$
      $(d_i, \tau_i, S_i) = \mathcal{W}(e_i, A_i+B_i)$
      $U_i = unify(S_i\rho_i \to \tau_i,\ S_i\sigma_{i-1})$
      $T_i = U_iS_i$
      $g_i = (T_ig_{i-1} \parallel T_iq_i \Rightarrow U_id_i)$
      $\sigma_i = T_i\sigma_{i-1}$
      $R_i = T_iR_{i-1}$
  return $\mathcal{D}(g_n, \sigma_n, R_n)$;

$\mathcal{W}((\textbf{spoly}\ x : \bigcap \alpha{:}D.\tau\ \textbf{as}\ key_1\ e_1 \mid \cdots \mid key_n \Rightarrow e_n), A) =$
  check that $\{key_1, \ldots, key_n\}$ cover D
  check that $\{key_1, \ldots, key_n\}$ are compatible
  $d_0 = \cdot$; $R_0 = Id$; $A_0 = A+\{x{:}\bigcap \alpha{:}D.\tau\}$
  for $i = 1$ to n:
    let $A_i = S_{i-1}A_{i-1}$
      $(e_i', \tau_i, S_i) = \mathcal{W}(e_i, A_i)$
      $\rho_i = \mathcal{TP}(D, key_i)$
      $U_i = unify(\tau_i,\ S_i\tau\ [\rho_i/\alpha])$
      if $FV(\rho_i) \subseteq dom(U_i)$ then **fail**
      $T_i = U_iS_i$
      $d_i = (T_id_{i-1} \mid \rho_i \Rrightarrow U_ie_i')$
      $R_i = T_iR_{i-1}$
  return $(\mu x.\ d_n, \bigcap \alpha{:}D.R_n\tau, R_n)$;

$\mathcal{W}$((**spoly** x :$\bigcap\varphi{:}\kappa\to D.\tau$ **as** $key_1\ e_1 \mid \cdots \mid key_n \Rightarrow e_n$), A) =

check that $\{key_1,\ldots,key_n\}$ cover $\kappa\to D$

check that $\{key_1,\ldots,key_n\}$ are compatible

$d_0 = \cdot$; $R_0 = Id$; $A_0 = A+\{x{:}\bigcap\varphi{:}\kappa\to D.\tau\}$

for i = 1 to n:

let $A_i = S_{i-1}A_{i-1}$

$(e_i',\tau_i,S_i) = \mathcal{W}(e_i,A_i)$

$\rho_i = \mathcal{TP}(\kappa\to D,\ key_i)$

$U_i = unify(\tau_i,\ S_i\tau\ [\rho_i/\varphi])$

if $FV(\rho_i)\subseteq dom(U_i)$ then **fail**

$T_i = U_iS_i$

$d_i = (T_id_{i-1} \mid \rho_i \twoheadrightarrow U_ie_i')$

$R_i = T_iR_{i-1}$

return $(\mu x.\ d_n,\ \bigcap s_v{:}\kappa\to D.\ R_n\tau,\ R_n)$;