Chiron Notation

William M. Farmer

24 May 2011

	Official	Compact	ASCII
PROPER EXPRESSIONS			
Operator	$(\mathsf{op}, o, k_1, \ldots, k_{n+1})$	$(o :: k_1, \ldots, k_{n+1})$	$(o$ `::` k_1`,...,`$k_{n+1})$
Operator application	$(\mathsf{op\text{-}app}, (\mathsf{op}, o, k_1, \ldots, k_{n+1}), e_1, \ldots, e_n)$	$(o :: k_1, \ldots, k_{n+1})(e_1, \ldots, e_n)$	$(o$ `::` k_1`,...,`$k_{n+1})(e_1$`,...,`$e_n)$
Constant	(con, o, k)	$[o :: k]$	`[`o `::` k`]`
Variable	$(\mathsf{var}, x, \alpha)$	$(x : \alpha)$	`(`x `:` α`)`
Type application	$(\mathsf{type\text{-}app}, \alpha, a)$	$\alpha(a)$	α`(`a`)`
Dependent function type	$(\mathsf{dep\text{-}fun\text{-}type}, (\mathsf{var}, x, \alpha), \beta)$	$(\Lambda\, x : \alpha\ .\ \beta)$	`(Lambda` x `:` α `.` β`)`
			`(`x `:` α `->` β`)`
Function application	$(\mathsf{fun\text{-}app}, f, a)$	$f(a)$	f`(`a`)`
Function abstraction	$(\mathsf{fun\text{-}abs}, (\mathsf{var}, x, \alpha), b)$	$(\lambda\, x : \alpha\ .\ b)$	`(lambda` x `:` α `.` b`)`
Conditional term	(if, A, b, c)	$\mathsf{if}(A, b, c)$	`if(`A`,`b`,`c`)`
Existential quantification	$(\mathsf{exists}, (\mathsf{var}, x, \alpha), B)$	$(\exists\, x : \alpha\ .\ B)$	`(exists` x `:` α `.` B`)`
Unique existential	$(\mathsf{uni\text{-}exists}, (\mathsf{var}, x, \alpha), B)$	$(\exists !\, x : \alpha\ .\ B)$	`(exists!` x `:` α `.` B`)`
Universal quantification	$(\mathsf{forall}, (\mathsf{var}, x, \alpha), B)$	$(\forall\, x : \alpha\ .\ B)$	`(forall` x `:` α `.` B`)`
Definite description	$(\mathsf{def\text{-}des}, (\mathsf{var}, x, \alpha), B)$	$(\iota\, x : \alpha\ .\ B)$	`(iota` x `:` α `.` B`)`
Indefinite description	$(\mathsf{indef\text{-}des}, (\mathsf{var}, x, \alpha), B)$	$(\epsilon\, x : \alpha\ .\ B)$	`(epsilon` x `:` α `.` B`)`
Set Construction	$(\mathsf{set\text{-}cons}, a_1, \ldots, a_n)$	$\{a_1, \ldots, a_n\}$	$\{a_1, \ldots, a_n\}$
List Construction	$(\mathsf{list\text{-}cons}, a_1, \ldots, a_n)$	$[a_1, \ldots, a_n]$	`[`$a_1, \ldots, a_n$`]`
Class abstraction	$(\mathsf{class\text{-}abs}, (\mathsf{var}, x, \alpha), B)$	$(\mathrm{C}\, x : \alpha\ .\ B)$	`(class` x `:` α `.` B`)`
Left type	$(\mathsf{left\text{-}type}, \alpha)$	$\mathsf{left}(\alpha)$	`left(`α`)`
Right type	$(\mathsf{right\text{-}type}, \alpha, a)$	$\mathsf{right}(\alpha, a)$	`right(`α`,`a`)`
Dependent type product	$(\mathsf{dep\text{-}type\text{-}prod}, (\mathsf{var}, x, \alpha), \beta)$	$(\otimes\, x : \alpha\ .\ \beta)$	`(*` x `:` α `.` β`)`
Dependent ordered pair	$(\mathsf{dep\text{-}ord\text{-}pair}, a, b)$	$\langle a, b \rangle$	`<`a`,`b`>`
Dependent head	$(\mathsf{dep\text{-}head}, a)$	$\mathsf{hd}(a, b)$	`hd(`a`,`b`)`
Dependent tail	$(\mathsf{dep\text{-}tail}, a)$	$\mathsf{tl}(a, b)$	`tl(`a`,`b`)`
Quotation	(quote, e)	$\ulcorner e \urcorner$	`|^e^|`
Quasiquotation	$(\mathsf{quasiquote}, (\mathsf{fun\text{-}app}, f, \lfloor a \rfloor))$	$\ulcorner f(\lfloor a \rfloor) \urcorner$	`|^`f`(|_`a`_|)^|`
Evaluation	(eval, a, k)	$[\![a]\!]_k$	`[|a|]_`k
Evaluation for types	$(\mathsf{eval}, a, \mathsf{type})$	$[\![a]\!]_{\mathrm{ty}}$	`[|a|]_ty`
Evaluation for terms	$(\mathsf{eval}, a, \mathsf{C})$	$[\![a]\!]_{\mathrm{te}}$	`[|a|]_te`
Evaluation for formulas	$(\mathsf{eval}, a, \mathsf{formula})$	$[\![a]\!]_{\mathrm{fo}}$	`[|a|]_fo`

	Official	Compact	ASCII
TYPES			
Set type	(con, set, type)	V	`V`
Class type	(con, class, type)	C	`C`
Type for expressions	(con, expr, type)	E	`E`
Type for symbols	(con, expr-sym, type)	E_{sy}	`E_sy`
Type for operator names	(con, expr-op-name, type)	E_{on}	`E_on`
Type for operators	(con, expr-op, type)	E_{op}	`E_op`
Type for types	(con, expr-type, type)	E_{ty}	`E_ty`
Type for terms	(con, expr-term, type)	E_{te}	`E_te`
Type for terms with type	(op-app, (op, expr-term-type, E_{ty}, type), a)	$\mathsf{E}^{a}_{\mathrm{te}}$	`E_te[`a`]`
Type for formulas	(con, expr-formula, type)	E_{fo}	`E_fo`
EQUALITIES			
Type equality	(op-app, (op, type-equal, type, type, formula), α, β)	$(\alpha =_{\mathrm{ty}} \beta)$	`(`α `=_ty` β`)`
Term equality in a type	(op-app, (op, term-equal, C, C, type, formula), a, b, α)	$(a =_{\alpha} b)$	`(`a `=_`α b`)`
Term equality	(op-app, (op, term-equal, C, C, type, formula), a, b, C)	$(a = b)$	`(`a `=` b`)`
Quasi-equality	(op-app, (op, quasi-equal, C, C, formula), a, b)	$(a \simeq b)$	`(`a `==` b`)`
Formula equality	(op-app, (op, formula-equal, formula, formula, formula), A, B)	$(A \equiv B)$	`(`A `iff` B`)`
LOGIC			
Truth	(con, true, formula)	T	`T`
Falsehood	(con, false, formula)	F	`F`
Negation	(op-app, (op, not, formula, formula), A)	$(\neg A)$	`not(`A`)`
Disjunction	(op-app, (op, or, formula, formula, formula), A, B)	$(A \vee B)$	`(`A `or` B`)`
Conjunction	(op-app, (op, and, formula, formula, formula), A, B)	$(A \wedge B)$	`(`A `and` B`)`
Implication	(op-app, (op, implies, formula, formula, formula), A, B)	$(A \supset B)$	`(`A `implies` B`)`
Definedness in a type	(op-app, (op, defined-in, C, type, formula), a, α)	$(a \downarrow \alpha)$	`#(`a`,`α`)`
Definedness	(op-app, (op, defined-in, C, type, formula), a, C)	$(a\downarrow)$	`#(`a`)`
Canonical undefined term	(con, undefined, C)	$\bot_{\mathsf{C}}$	`undef`
Canonical empty type	(con, empty-type, type)	∇	`empty-type`
Type order	(op-app, (op, type-le, type, type, formula), α, β)	$(\alpha \ll \beta)$	`(`α `<<` β`)`
Conditional type	(op-app, (op, if-type, formula, type, type, type), A, β, γ)	$\mathsf{if}(A, \beta, \gamma)$	`if(`A`,`β`,`γ`)`
Conditional formula	(op-app, (op, if-formula, formula, formula, formula, formula), A, B, C)	$\mathsf{if}(A, B, C)$	`if(`A`,`B`,`C`)`
Simple function type	(op-app, (op, sim-fun-type, type, type, type), α, β)	$(\alpha \to \beta)$	`(`α `->` β`)`

	Official	Compact	ASCII
SET THEORY			
Class membership	(op-app, (op, in, V, C, formula), a, b)	$(a \in b)$	`(`a `in` b`)`
Empty set	(con, empty-set, V)	$\emptyset$	`empty-set`
Universal class	(con, universal-class, C)	U	`universe`
Pair	(op-app, (op, pair, V, V, V), a, b)	$\{a, b\}$	`{`a`,`b`}`
Ordered pair	(op-app, (op, ord-pair, V, V, V), a, b)	$\langle a, b \rangle$	`<`a`,`b`>`
Subclass	(op-app, (op, subclass, C, C, formula), a, b)	$a \subseteq b$	`(`a `subcl` b`)`
Union	(op-app, (op, union, C, C, C), a, b)	$a \cup b$	`(`a `union` b`)`
Intersection	(op-app, (op, intersection, C, C, C), a, b)	$a \cap b$	`(`a `inters` b`)`
Complement	(op-app, (op, complement, C, C), a)	$\overline{a}$	`compl(`a`)`
Head	(op-app, (op, head, V, V), a)	$\mathsf{hd}(a)$	`hd(`a`)`
Tail	(op-app, (op, tail, V, V), a)	$\mathsf{tl}(a)$	`tl(`a`)`
Type product	(op-app, (op, type-prod, type, type, type), α, β)	$(\alpha \times \beta)$	`(`α `*` β`)`
Type sum	(op-app, (op, type-sum, type, type, type), α, β)	$(\alpha + \beta)$	`(`α `+` β`)`
Type to Term	(op-app, (op, type-to-term, type, C), α)	$\mathsf{term}(\alpha)$	`term(`α`)`
Term to Type	(op-app, (op, term-to-type, C, type), α)	$\mathsf{type}(a)$	`type(`a`)`