

# The Virtues of Semi-Explicit Polymorphism

Frank Emrich

The University of Edinburgh, UK

frank.emrich@ed.ac.uk

Sam Lindley

Heriot-Watt University, UK

s.lindley@hw.ac.uk

Jan Stolarek

The University of Edinburgh, UK

jan.stolarek@ed.ac.uk

## Abstract

There are two standard ways of specifying a type system for ML: an orthogonal presentation and a syntax-directed presentation. The former allows implicit generalisation and instantiation anywhere in a program, and is thus not syntax-directed. The latter fuses generalisation with let-bindings and instantiation with variables, and is thus non-orthogonal. By introducing explicit syntax for generalisation and instantiation, that is, semi-explicit polymorphism, we obtain a presentation of Explicit ML, a mild variant of ML, which is both orthogonal and syntax-directed. Moreover, we recover the usual implicit version of ML as syntactic sugar.

FreezeML is a small extension of ML providing first-class polymorphism and sound and complete type inference of principal types, whose typing rules are non-orthogonal. We show that Explicit ML extends naturally to Explicit FreezeML, an orthogonal syntax-directed presentation of an explicit variant of FreezeML. We recover the usual implicit version of FreezeML as syntactic sugar. Explicit FreezeML is a conservative extension of both Explicit ML and System F.

## 1 Introduction

The design of ML is motivated by a desire to write polymorphic programs without laboriously spelling out details of type abstraction and type application. A remarkable feature of ML is that, due to its restricted form of polymorphism, it is unnecessary to write any polymorphism, or indeed any types, at all. The usual orthogonal (or *declarative*) presentation of ML [2] exploits this property by not even providing syntax to mark where generalisation and instantiation occur. The usual syntax-directed presentation of ML [1] takes advantage of the fact that it is sufficient to only generalise let-bindings and only (and always) instantiate variables.

As ML programmers we, the authors, prefer the determinism of the syntax-directed presentation, and would argue that it is closer to the intuitive model we use in practice when writing and reasoning about ML programs. However, the syntax-directed presentation is non-orthogonal exactly because it fuses generalisation with let-binding and instantiation with variables. By adding explicit syntax for generalisation and instantiation, we obtain an orthogonal and syntax-directed language, *Explicit ML*. Moreover, we recover the usual implicit version of ML as syntactic sugar.

Explicit ML is no more expressive than implicit ML, and on the face of it may seem like a superficial conceptual improvement. However, as we shall see, where it really shines is when we extend ML with first-class polymorphism.

The *prenex polymorphism* of ML only allows top-level quantifiers and only allows quantifiers to be instantiated with monomorphic types. *FreezeML* [4] is a small extension of ML providing first-class polymorphism and sound and complete type inference of principal types. It is part of a large design space of systems bridging the gap between tractable type inference and first-class polymorphism [5–10, 12–16]. FreezeML adds optional type annotations on bound variables and a construct for *freezing* variables, preventing them from being implicitly instantiated. Whilst the previous formulation of FreezeML is not orthogonal, we introduce *Explicit FreezeML*, a natural extension of Explicit ML, which is both orthogonal and syntax-directed. We may recover FreezeML as syntactic sugar for Explicit FreezeML.

We distinguish three forms of polymorphism.

<b>implicit</b>	implicit generalisation + instantiation
<b>semi-explicit</b>	explicit generalisation + instantiation
<b>explicit</b>	type abstraction + type application

Prior systems with semi-explicit polymorphism include IFX [10], Poly-ML [5], and QML [12]. They distinguish ML-like type schemes and System F-style explicit polymorphism, whereas (Explicit) FreezeML has only System F types.

The perspective we take in this work is that Explicit ML (or Explicit FreezeML) is the programming language, and ML (or FreezeML) is merely syntactic sugar. Figure 1 illustrates the path from syntactic sugar (first column) to programming language (second column) to core language (third column).

The rest of this extended abstract outlines the design of Explicit ML and Explicit FreezeML, desugaring rules, and a succinct equational theory that dictates elaboration to System F. Full details appear in the appendix.

## 2 Explicit ML

We let  $S, T$  range over monomorphic types and  $E, F$  range over type schemes. Typing judgements have the form  $\Delta; \Gamma \vdash M : E$ , stating that term  $M$  has type scheme  $E$  in type context  $\Delta$  (a sequence of type variables ranged over by  $a, b$ ) and term context  $\Gamma$ . (Traditional presentations of ML often elide which type variables  $\Delta$  are in scope; we prefer to track these explicitly.)

**Generalisation.** In ML, implicit generalisation is introduced by the following rule.

$$\frac{\Delta, \Delta'; \Gamma \vdash M : S}{\Delta; \Gamma \vdash M : \forall \Delta'. S}$$

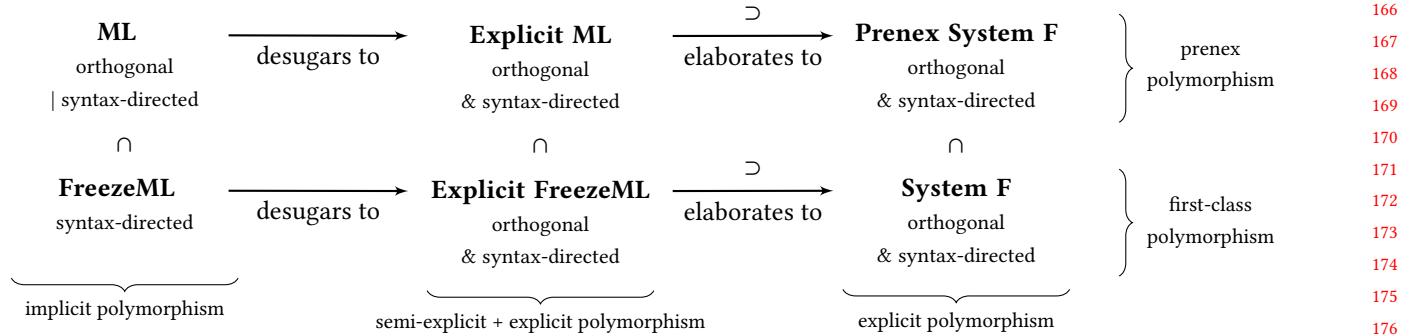


Figure 1. Desugaring and Elaboration of ML and FreezeML

It allows terms to be given arbitrarily general types. For instance, the generalised identity function  $\lambda x.x$  may be typed as  $\text{Int} \rightarrow \text{Int}$ , as  $\forall a.a \rightarrow a$ , as  $\forall a.b.(a \rightarrow b) \rightarrow (a \rightarrow b)$ , or as infinitely many other types. As it will become necessary later, we adopt a stricter notion of generalisation.

$$\frac{\text{I-GEN} \quad \Delta, \Delta'; \Gamma \vdash M : S \quad \text{principal}(\Delta, \Gamma, M, \Delta', S)}{\Delta; \Gamma \vdash M : \forall \Delta'.S}$$

The principal constraint (Appendix E.2) ensures that generalisation yields the unique most general type. For instance, the generalised identity function  $\lambda x.x$  may now only be typed as  $\forall a.a \rightarrow a$ . Explicit ML adopts a variant of I-GEN in which generalisation is explicit in the syntax of terms.

$$\frac{\text{GEN} \quad \Delta, \Delta'; \Gamma \vdash M : S \quad \text{principal}(\Delta, \Gamma, M, \Delta', S)}{\Delta; \Gamma \vdash \Lambda \bullet . M : \forall \Delta'.S}$$

The astute reader may be concerned about circularity in the definition of principality. Fortunately, we may give a mutually inductive definition of typing and principality by indexing both judgements by the untyped term [3].

**Instantiation.** The implicit instantiation rule of ML, substitutes monomorphic types for the body of a term.

$$\frac{\text{I-INST} \quad \Delta; \Gamma \vdash M : \forall \Delta'.S \quad \Delta \vdash \sigma : \Delta' \Rightarrow .}{\Delta; \Gamma \vdash M : \sigma(S)}$$

The judgement  $\Delta \vdash \sigma : \Delta' \Rightarrow \Delta''$  defines a type instantiation  $\sigma$  mapping type variables in  $(\Delta, \Delta')$  to types with free type variables in  $(\Delta, \Delta'')$ , such that  $\sigma(a) = a$  for every  $a \in \Delta$ .

Explicit ML adopts a variation of I-INST in which instantiation is explicit in the syntax of terms.

$$\frac{\text{INST} \quad \Delta; \Gamma \vdash M : \forall \Delta'.S \quad \Delta \vdash \sigma : \Delta' \Rightarrow .}{\Delta; \Gamma \vdash M \bullet : \sigma(S)}$$

**Variables and let-binding.** We write variables as  $[x]$  and let-binding as  $\text{let } [x] = M \text{ in } N$ . We say that such variables are *frozen* as they are not implicitly instantiated.

Similarly, we say that such let-bindings are *frozen* as they do not implicitly generalise  $M$ .

We now define implicit instantiation of variables and implicit generalisation of let-bindings as syntactic sugar.

$$\begin{aligned} x &\equiv [x]\bullet \\ \text{let } x = M \text{ in } N &\equiv \text{let } [x] = \Lambda \bullet . M \text{ in } N \end{aligned}$$

## 2.1 Explicit Polymorphism

In addition to the semi-explicit polymorphism we have already seen, we also include fully explicit polymorphism in Explicit ML. This requires a little care. Suppose we allow explicit type abstraction. Now consider the term  $\Lambda a.\lambda x.x$ . It is not immediately clear whether this term should have principal type  $\forall a.a \rightarrow a$  or  $\forall a.b \rightarrow b$ . Exactly the same problem occurs with the term:  $\Lambda a.\text{id}$  where  $\text{id} : \forall a.a \rightarrow a$ .

We adopt an approach that ensures that the body of a type abstraction has a unique typing. We do so by dividing the syntax of Explicit ML terms into two classes.

$$\begin{array}{ll} \text{ITerm} \ni & \text{MTerm} \ni \\ J, I ::= [x] & M, N ::= [x] \\ | \lambda(x : S).I \mid I N & | \lambda(x : S).M \mid M N \\ | \Lambda a.I \mid I S & | \Lambda a.I \mid M S \\ | \text{let } [x] = I \text{ in } J & | \text{let } [x] = M \text{ in } N \\ | \Lambda \bullet . M & | \lambda x.M \\ & | \Lambda \bullet . M \\ & | M \bullet \end{array}$$

The ITerm class consists of Prenex System F extended with (frozen, i.e., non-generalising) let-binding and generalisation. The body of a generalisation need not be an ITerm as generalisation always yields the unique most general type. Similarly, the argument of a function application need not be an ITerm as the type of a function uniquely determines its return type. The MTerm class adds unannotated lambdas and implicit instantiation, these being the only two sources of non-determinism in type inference.

Explicit ML subsumes both Prenex System F and ML: the former directly and the latter via syntactic sugar.

### 3 Explicit FreezeML

The extension of Explicit ML to Explicit FreezeML is modest. Types may now be fully polymorphic. We let  $A, B$  range over System F types. Some care must be taken to manage the separation between monomorphic and polymorphic types. To control where polymorphic instantiation takes place Explicit FreezeML adds a third class of terms.

$$\begin{array}{l} \text{ITerm } \ni I, J ::= [x] \\ \quad | \lambda(x : A).I \mid IQ \\ \quad | \Lambda a.I \mid IA \\ \quad | \text{let } [x] = I \text{ in } J \\ \quad | \Lambda \bullet .P \\ \\ \text{MTerm } \ni M, N ::= [x] \\ \quad | \lambda(x : A).M \mid MQ \\ \quad | \Lambda a.I \mid MA \\ \quad | \text{let } [x] = M \text{ in } N \\ \quad | \lambda x.M \\ \quad | \Lambda \bullet .P \\ \quad | M \bullet \\ \\ \text{PTerm } \ni P, Q ::= [x] \\ \quad | \lambda(x : A).P \mid PQ \\ \quad | \Lambda a.I \mid PA \\ \quad | \text{let } [x] = M \text{ in } Q \\ \quad | \lambda x.P \\ \quad | \Lambda \bullet .P \\ \quad | P \bullet \\ \quad | P \star \end{array}$$

The PTerm class extends MTerm with a polymorphic instantiation operator  $P\star$ . The key place where it is important to restrict terms to use monomorphic instantiation is in let-bindings. This restriction prevents “guessing polymorphism”, keeping type inference tractable [11, 17]. For the same reason, the typing rule for unannotated lambda abstractions is restricted to monomorphic argument types. The Explicit FreezeML typing judgement has the form  $\Delta; \Gamma \vdash P : A$ .

We now define the implicit instantiation of variables and implicit generalisation of let-bindings as syntactic sugar.

$$\begin{array}{lcl} x & \equiv & [x]\star \\ \text{let } x = P \text{ in } Q & \equiv & \text{let } [x] = \Lambda \bullet .P \text{ in } Q \end{array}$$

Moreover, using intermediate syntactic sugar for type-annotated terms and in turn type-annotated generalisation, we define the type-annotated variant of generalising let from FreezeML as syntactic sugar.

$$\begin{array}{lcl} (P : A) & \equiv & (\lambda(x : A).[x])P \\ (\Lambda \bullet .P : \forall \Delta.G) & \equiv & \Lambda \Delta.(P : G) \\ \text{let } (x : A) = P \text{ in } Q & \equiv & (\lambda(x : A).Q)(\Lambda \bullet .P : A) \end{array}$$

Here  $G$  ranges over *guarded types*, that is, types whose outermost type constructor is not  $\forall$ . We also define syntactic sugar for non-generalising variants of let in which the let-binding is not syntactically restricted to be an MTerm.

$$\begin{array}{lcl} \text{let}' x = P \text{ in } Q & \equiv & \text{let } [x] = (\Lambda \bullet .P)\bullet \text{ in } Q \\ \text{let}' (x : A) = P \text{ in } Q & \equiv & (\lambda(x : A).Q)P \end{array}$$

In the unannotated case the term  $(\Lambda \bullet .P)\bullet$  has the effect of ensuring that all instantiations inside  $P$  are monomorphic. We can now implement the value restriction [18] by deciding

whether or not to generalise a let-bound term depending on whether it is a syntactic value or not (Appendix G).

Explicit FreezeML subsumes both System F and FreezeML: the former directly and the latter via syntactic sugar.

The type inference algorithm for Explicit FreezeML is a minor adaptation of the one for FreezeML [4], which is itself a routine extension of algorithm W [2].

**Equational Reasoning.** The equivalence  $P \simeq Q$  on terms  $P$  and  $Q$  is defined only when  $P$  and  $Q$  have the same type in the same context (i.e.,  $\Delta; \Gamma \vdash P : A$  and  $\Delta; \Gamma \vdash Q : A$ ). The following rules are the usual  $\beta$  and  $\eta$ -rules of System F.

$$\begin{array}{ll} \beta\text{-rules} & (\lambda(x : A).P)Q \simeq P[Q/[x]] \\ & (\Lambda a.I)A \simeq I[A/a] \\ \eta\text{-rules} & \lambda(x : A).P[x] \simeq P \\ & \Lambda a.Ia \simeq I \end{array}$$

The following rules elaborate the additional constructs of Explicit FreezeML into plain System F terms.

$$\begin{array}{ll} \text{let } [x] = M \text{ in } Q & \simeq (\lambda(x : A).Q)M \\ \lambda x.P & \simeq \lambda(x : S).P \\ \Lambda \bullet .I & \simeq \Lambda \Delta.I \\ P \bullet & \simeq PS_1 \dots S_n \\ P \star & \simeq PA_1 \dots A_n \end{array}$$

Let bindings and unannotated lambdas are expressible using type-annotated lambda abstractions. The last three rules witness the correspondence between generalisation and type abstraction and between instantiation and type application. The third rule applies only once the body of a generalisation has been elaborated. The translation in Appendix E.4 lifts the elaboration rules to a translation on derivations and in so doing proves that we can systematically apply them left-to-right to elaborate to System F.

## 4 Conclusions and Future Work

FreezeML is a pragmatic extension of ML with first-class polymorphism. In Explicit FreezeML, by making generalisation and instantiation explicit, we have obtained an orthogonal presentation of FreezeML. More ad hoc aspects of FreezeML arise as syntactic sugar on top of Explicit FreezeML.

More sophisticated approaches to first-class polymorphism use heuristics [8, 13, 14] to avoid explicitly marking generalisation and instantiation. We plan to investigate the extent to which we can capture such heuristics via syntactic sugar or lightweight typing extensions on top of Explicit FreezeML. We also plan to extend Explicit FreezeML to support  $F\omega$  and to adapt Explicit FreezeML to account for features such as typing constraints and bidirectional typing.

Quite apart from first-class polymorphism, we believe that ad hoc conveniences such as implicit generalisation and instantiation are best defined as syntactic sugar. The benefits to designing orthogonal languages with syntax-directed typing rules are both conceptual and practical.

## 331 References

- 332 [1] Dominique Clément, Joëlle Despeyroux, Thierry Despeyroux, and  
333 Gilles Kahn. 1986. A Simple Applicative Language: Mini-ML. In *LISP*  
334 and *Functional Programming*. ACM, 13–27.
- 335 [2] Luís Damas and Robin Milner. 1982. Principal Type-Schemes for  
336 Functional Programs. In *POPL*. ACM Press, 207–212.
- 337 [3] Frank Emrich, Sam Lindley, Jan Stolarek, James Cheney, and Jonathan  
338 Coates. 2019. *FreezeML: Complete and Easy Type Inference for First-Class*  
339 *Polymorphism (extended version)*. Technical Report. arXiv:2004.00396.
- 340 [4] Frank Emrich, Sam Lindley, Jan Stolarek, James Cheney, and Jonathan  
341 Coates. 2020. FreezeML: Complete and Easy Type Inference for First-  
342 Class Polymorphism. In *PLDI*. ACM. <https://arxiv.org/abs/2004.00396>.  
343 To appear.
- 344 [5] Jacques Garrigue and Didier Rémy. 1999. Semi-Explicit First-Class  
345 Polymorphism for ML. *Inf. Comput.* 155, 1-2 (1999), 134–169.
- 346 [6] Didier Le Botlan and Didier Rémy. 2003. ML<sup>F</sup>: raising ML to the power  
347 of System F. In *ICFP*. ACM, 27–38.
- 348 [7] Daan Leijen. 2007. A type directed translation of MLF to system F. In  
349 *ICFP*. ACM, 111–122.
- 350 [8] Daan Leijen. 2008. HMF: simple type inference for first-class polymor-  
351 phism. In *ICFP*. ACM, 283–294.
- 352 [9] Daan Leijen. 2009. Flexible types: robust type inference for first-class  
353 polymorphism. In *POPL*. ACM, 66–77.
- 354
- 355
- 356
- 357
- 358
- 359
- 360
- 361
- 362
- 363
- 364
- 365
- 366
- 367
- 368
- 369
- 370
- 371
- 372
- 373
- 374
- 375
- 376
- 377
- 378
- 379
- 380
- 381
- 382
- 383
- 384
- 385
- [10] James O’Toole and David K. Gifford. 1989. Type Reconstruction with  
386 First-Class Polymorphic Values. In *PLDI*. ACM, 207–217. <https://doi.org/10.1145/73141.74836>
- [11] Frank Pfenning. 1993. On the Undecidability of Partial Polymorphic  
388 Type Reconstruction. *Fundam. Inform.* 19, 1/2 (1993), 185–199.
- [12] Claudio V. Russo and Dimitrios Vytiniotis. 2009. QML: Explicit First-  
389 class Polymorphism for ML. In *ML*. ACM, 3–14.
- [13] Alejandro Serrano, Jurriaan Hage, Simon Peyton Jones, and Dimitrios  
390 Vytiniotis. 2020. A quick look at impredicativity. In *ICFP*. ACM. Con-  
ditionally accepted.
- [14] Alejandro Serrano, Jurriaan Hage, Dimitrios Vytiniotis, and Simon  
391 Peyton Jones. 2018. Guarded impredicative polymorphism. In *PLDI*.  
392 ACM, 783–796.
- [15] Dimitrios Vytiniotis, Stephanie Weirich, and Simon L. Peyton Jones.  
393 2006. Boxy types: inference for higher-rank types and impredicativity.  
In *ICFP*. ACM, 251–262.
- [16] Dimitrios Vytiniotis, Stephanie Weirich, and Simon L. Peyton Jones.  
394 2008. FPH: first-class polymorphism for Haskell. In *ICFP*. ACM, 295–  
395 306.
- [17] J. B. Wells. 1994. Typability and Type-Checking in the Second-Order  
396 lambda-Calculus are Equivalent and Undecidable. In *LICS*. IEEE Com-  
puter Society, 176–185.
- [18] Andrew K. Wright. 1995. Simple Imperative Polymorphism. *Lisp and  
397 Symbolic Computation* 8, 4 (1995), 343–355.
- 398
- 399
- 400
- 401
- 402
- 403
- 404
- 405
- 406
- 407
- 408
- 409
- 410
- 411
- 412
- 413
- 414
- 415
- 416
- 417
- 418
- 419
- 420
- 421
- 422
- 423
- 424
- 425
- 426
- 427
- 428
- 429
- 430
- 431
- 432
- 433
- 434
- 435
- 436
- 437
- 438
- 439
- 440

441	<b>A Prenex System F</b>	496	
442	<b>A.1 Syntax of Prenex System F</b>	497	
443		498	
444		499	
445	Type Variables	$a, b, c$	500
446	Type Constructors	$D ::= \text{Int} \mid \text{List} \mid \rightarrow \mid \times \mid \dots$	501
447	Monotypes	$S, T ::= a \mid D \bar{S}$	502
448	Type Schemes	$E, F ::= \forall \bar{a}. S$	503
449	Type Contexts	$\Delta ::= \cdot \mid \Delta, a$	504
450	Term Contexts	$\Gamma ::= \cdot \mid \Gamma, x : E$	505
451	Term Variables	$x, y, z$	506
452	Terms	$M, N ::= [x] \mid \lambda(x : S). M \mid M N \mid \Lambda a. M \mid M S$	507
453			508
454	<b>A.2 Type System of Prenex System F</b>	509	
455	<i>Well-formed monotypes / type schemes.</i>	$\boxed{\Delta \vdash E \text{ ok}}$	510
456			511
457			512
458			513
459	$a \in \Delta$	$\Delta \vdash a \text{ ok}$	514
460			515
461		$\frac{\text{arity}(D) = n}{\Delta \vdash E_1 \text{ ok} \dots \Delta \vdash E_n \text{ ok}} \Delta \vdash D \bar{E} \text{ ok}$	516
462	<i>Typing.</i>	$\boxed{\Delta; \Gamma \vdash M : E}$	517
463			518
464			519
465			520
466	$\text{VAR}$	$\frac{\text{APP}}{\Delta; \Gamma \vdash x : E}$	521
467		$\Delta; \Gamma \vdash M : S \rightarrow T$	
468		$\Delta; \Gamma \vdash N : S$	
469		$\Delta; \Gamma \vdash M N : T$	
470	$\text{TyLAM}$	$\frac{\text{LAM}}{\Delta; \Gamma \vdash \Lambda a. M : \forall a. E}$	
471		$\Delta; a; \Gamma \vdash M : E$	
472		$\Delta; \Gamma, x : S \vdash M : T$	
473		$\Delta; \Gamma \vdash \lambda(x : S). M : S \rightarrow T$	
474	$\text{TYAPP}$	$\frac{\text{TYAPP}}{\Delta; \Gamma \vdash M : \forall a. E}$	522
475		$\Delta; \Gamma \vdash M S : E[S/a]$	523
476			524
477			525
478			526
479	$\beta$ -rules	$(\lambda(x : S). M) N \simeq M[N/[x]]$	531
480		$(\Lambda a. M) S \simeq M[S/a]$	532
481	$\eta$ -rules	$\lambda(x : S). M [x] \simeq M$	534
482		$\Lambda a. M a \simeq M$	535
483	<b>B Explicit ML</b>	538	
484	<b>B.1 Syntax of Explicit ML</b>	539	
485	<i>Types.</i>	$\boxed{\cdot}$	540
486			541
487			542
488	Type Variables	$a, b, c$	543
489	Type Constructors	$D ::= \text{Int} \mid \text{List} \mid \rightarrow \mid \times \mid \dots$	544
490	Monotypes	$S, T ::= a \mid D \bar{S}$	545
491	Type Schemes	$E, F ::= \forall \bar{a}. S$	546
492	Type Instantiation	$\sigma ::= \emptyset \mid \sigma[a \mapsto S]$	547
493	Type Contexts	$\Delta ::= \cdot \mid \Delta, a$	548
494	Term Contexts	$\Gamma ::= \cdot \mid \Gamma, x : E$	549
495			550

551	<b>Terms.</b>	606
552		607
553	$\text{ITerm} \ni I, J ::= [x]$	608
554	$\lambda(x : S).I \mid IN$	609
555	$\Lambda a.I \mid IS$	610
556	$\text{let } [x] = I \text{ in } J$	611
557	$\Lambda\bullet.M$	612
558		613
559		614
560		615
561	<b>B.2 Type System of Explicit ML</b>	616
562	<i>Well-formed monotypes / type schemes.</i> $\boxed{\Delta \vdash E \text{ ok}}$	617
563		618
564	$\frac{a \in \Delta}{\Delta \vdash a \text{ ok}}$	619
565	$\frac{\text{arity}(D) = n \quad \Delta \vdash E_1 \text{ ok} \quad \dots \quad \Delta \vdash E_n \text{ ok}}{\Delta \vdash D \bar{E} \text{ ok}}$	620
566		621
567		622
568	<i>Instantiation.</i> $\boxed{\Delta \vdash \sigma : \Delta' \Rightarrow \Delta''}$	623
569		624
570		625
571		626
572	$\frac{}{\Delta \vdash \emptyset : \cdot \Rightarrow \Delta'}$	627
573		628
574	<i>Principality.</i> $\boxed{\text{principal}(\Delta, \Gamma, M, \Delta', E)}$	629
575		630
576		631
577		632
578	$\text{principal}(\Delta, \Gamma, M, \Delta', E') =$	633
579	$\Delta' = \text{ftv}(E') - \Delta$ and $\Delta, \Delta'; \Gamma \vdash M : E'$ and	634
580	(for all $\Delta'', E''$   if $\Delta'' = \text{ftv}(E'') - \Delta$ and	635
581	$\Delta, \Delta''; \Gamma \vdash M : E''$	636
582	then there exists $\sigma$ such that	637
583	$\Delta \vdash \sigma : \Delta' \Rightarrow \Delta''$ and $\sigma(E') = E''$	638
584		639
585	<i>Typing.</i> $\boxed{\Delta; \Gamma \vdash M : E}$	640
586		641
587		642
588	$\text{VAR}$	643
589	$x : E \in \Gamma$	
590	$\frac{}{\Delta; \Gamma \vdash [x] : E}$	644
591		645
592	$\text{LAM}$	646
593	$\Delta; \Gamma, x : S \vdash M : T$	
594	$\frac{}{\Delta; \Gamma \vdash \lambda(x : S).M : S \rightarrow T}$	647
595		648
596	$\text{APP}$	649
597	$\Delta; \Gamma \vdash M : S \rightarrow T$	
598	$\Delta; \Gamma \vdash N : S$	650
599	$\frac{}{\Delta; \Gamma \vdash MN : T}$	
600		651
601	$\text{TYLAM}$	652
602	$\Delta; \Gamma \vdash I : F$	
603	$\Delta, a; \Gamma \vdash \Lambda a.I : \forall a.F$	653
604	$\frac{}{\Delta; \Gamma \vdash \Lambda a.I : \forall a.F}$	654
605		655
606	$\text{TYAPP}$	656
607	$\Delta; \Gamma \vdash M : \forall a.F$	
608	$\frac{}{\Delta; \Gamma \vdash MS : F[S/a]}$	657
609		658
610	$\text{LET}$	659
611	$\Delta; \Gamma \vdash M : E \quad \Delta; \Gamma, x : E \vdash N : T$	
612	$\frac{}{\Delta; \Gamma \vdash \text{let } [x] = M \text{ in } N : T}$	660
613	$\text{U-LAM}$	
614	$\Delta; \Gamma, x : S \vdash M : T$	651
615	$\frac{}{\Delta; \Gamma \vdash \lambda x.M : S \rightarrow T}$	652
616	$\text{MONOINST}$	653
617	$\Delta; \Gamma \vdash M : \forall \Delta'.S$	
618	$\Delta \vdash \sigma : \Delta' \Rightarrow \cdot$	654
619	$\frac{}{\Delta; \Gamma \vdash M\bullet : \sigma(S)}$	655
620		656
621	<b>B.3 Equational Rules of Explicit ML</b>	657
622	As in Section 3 the equivalence $M \simeq N$ on terms $M$ and $N$ is defined only when $M$ and $N$ have the same type in the same context (i.e., $\Delta; \Gamma \vdash M : E$ and $\Delta; \Gamma \vdash N : E$ ).	658
623		659
624		660

661					716
662	$\beta$ -rules	$(\lambda(x : S).M) N \simeq M[N/\lceil x \rceil]$			717
663		$(\Lambda a.I) S \simeq I[S/a]$			718
664					719
665	$\eta$ -rules	$\lambda(x : S).M \lceil x \rceil \simeq M$			720
666		$\Lambda a.I a \simeq I$			721
667					722
668	elaboration rules	$\text{let } [x] = M \text{ in } N \simeq (\lambda(x : E).N) M$			723
669		$\lambda x.M \simeq \lambda(x : S).M$			724
670		$\Lambda \bullet.I \simeq \Lambda \Delta.I$			725
671		$M \bullet \simeq M S_1 \dots S_n$			726

## B.4 Translation from Explicit ML to Prenex System F

672					727
673					728
674	$\left[ \frac{x : E \in \Gamma}{\Delta; \Gamma \vdash [x] : E} \right] = x$	$\left[ \frac{\Delta; \Gamma, x : A \vdash M : T}{\Delta; \Gamma \vdash \lambda(x : S).M : S \rightarrow T} \right] = \lambda(x : S).\llbracket M \rrbracket$	$\left[ \frac{\Delta; \Gamma \vdash M : S \rightarrow T \quad \Delta; \Gamma \vdash N : S}{\Delta; \Gamma \vdash M N : T} \right] = \llbracket M \rrbracket \llbracket N \rrbracket$		729
675					730
676					731
677	$\left[ \frac{\Delta, a; \Gamma \vdash I : E}{\Delta; \Gamma \vdash \Lambda a.I : \forall a.E} \right] = \Lambda a.\llbracket I \rrbracket$	$\left[ \frac{\Delta; \Gamma \vdash M : \forall a.E}{\Delta; \Gamma \vdash M S : E[S/a]} \right] = \llbracket M \rrbracket S$			732
678					733
679					734
680	$\left[ \frac{\Delta; \Gamma \vdash M : E \quad \Delta; \Gamma, x : E \vdash N : F}{\Delta; \Gamma \vdash \text{let } [x] = M \text{ in } N : F} \right] = (\lambda(x : E).\llbracket N \rrbracket) \llbracket M \rrbracket$	$\left[ \frac{\Delta; \Gamma, x : S \vdash M : T}{\Delta; \Gamma \vdash \lambda x.M : S \rightarrow T} \right] = \lambda(x : S).\llbracket M \rrbracket$			735
681					736
682					737
683					738
684	$\left[ \frac{\Delta, \Delta'; \Gamma \vdash M : E' \quad \text{principal}(\Delta, \Gamma, M, \Delta', E')}{\Delta; \Gamma \vdash \Lambda \bullet.M : \forall \Delta'.E'} \right] = \Lambda \Delta'.\llbracket M \rrbracket$	$\left[ \frac{\Delta; \Gamma \vdash M : \forall \Delta'.S \quad \Delta \vdash \sigma : \Delta' \Rightarrow \cdot}{\Delta; \Gamma \vdash M \bullet : \sigma(S)} \right] = \llbracket M \rrbracket \sigma(\Delta')$			739
685					740
686					741
687					742

## C ML

### C.1 Syntax of ML

#### Types.

692	Type Variables	$a, b, c$		747
693	Type Constructors	$D ::= \text{Int} \mid \text{List} \mid \rightarrow \mid \times \mid \dots$		748
694	Monotypes	$S, T ::= a \mid D \bar{S}$		749
695	Type Schemes	$E, F ::= \forall a.S$		750
696	Type Instantiation	$\sigma ::= \emptyset \mid \sigma[a \mapsto S]$		751
697	Type Contexts	$\Delta ::= \cdot \mid \Delta, a$		752
698	Term Contexts	$\Gamma ::= \cdot \mid \Gamma, x : E$		753
699				754

#### Terms.

700	$M, N ::= x$			755
701	$  \lambda x.M \mid MN$			756
702	$  \text{let } x = M \text{ in } N$			757
703				758

### C.2 Type System of ML

Well-formed monotypes / type schemes.  $\boxed{\Delta \vdash E \text{ ok}}$

707	$a \in \Delta$	$\frac{\text{arity}(D) = n}{\Delta \vdash a \text{ ok}}$	$\frac{\Delta \vdash S_1 \text{ ok} \quad \dots \quad \Delta \vdash S_n \text{ ok}}{\Delta \vdash D \bar{S} \text{ ok}}$	$\frac{\Delta, a \vdash E \text{ ok}}{\Delta \vdash \forall a.E \text{ ok}}$	762
708					763
709					764

Instantiation.  $\boxed{\Delta \vdash \sigma : \Delta' \Rightarrow \Delta''}$

710	$\frac{\Delta \vdash \sigma : \Delta' \Rightarrow \Delta'' \quad \Delta, \Delta'' \vdash S \text{ ok}}{\Delta \vdash \sigma[a \mapsto S] : (\Delta', a) \Rightarrow \Delta''}$	765
711		766
712		767
713		768
714		769
715		770

771	<b>Orthogonal Typing Judgement.</b>	$\boxed{\Delta; \Gamma \vdash M : E}$	826
772			827
773			828
774	VAR		829
775	$x : E \in \Gamma$	$\frac{\text{U-LAM}}{\Delta; \Gamma, x : S \vdash M : T}$	830
776		$\frac{}{\Delta; \Gamma \vdash \lambda x. M : S \rightarrow T}$	831
777			832
778	LET		833
779	$\Delta; \Gamma \vdash M : E$	I-GEN-LAX	834
780	$\Delta; \Gamma, x : E \vdash N : T$	$\frac{\Delta, \Delta'; \Gamma \vdash M : S}{\Delta; \Gamma \vdash M : \forall \Delta'. S}$	835
781			836
782	$\Delta; \Gamma \vdash \text{let } x = M \text{ in } N : T$	I-INST	837
783	<b>Syntax-directed Typing Judgement.</b>	$\boxed{\Delta; \Gamma \vdash M : S}$	838
784			839
785			840
786	VARINST		841
787	$x : \forall \Delta'. S \in \Gamma \quad \Delta \vdash \sigma : \Delta' \Rightarrow \cdot$	$\frac{\text{U-LAM}}{\Delta; \Gamma, x : S \vdash M : T}$	842
788		$\frac{}{\Delta; \Gamma \vdash \lambda x. M : S \rightarrow T}$	843
789			844
790	APP		845
791	$\Delta; \Gamma \vdash M : S \rightarrow T$	LETGEN	846
792	$\Delta; \Gamma \vdash N : S$	$\frac{\Delta' = \text{ftv}(S) - \Delta \quad \Delta, \Delta'; \Gamma \vdash M : S}{E = \forall \Delta'. S \quad \Delta; \Gamma, x : E \vdash N : T}$	847
793		$\frac{}{\Delta; \Gamma \vdash \text{let } x = M \text{ in } N : T}$	848
794			849
795	C.3 Desugaring from ML to Explicit ML		850
796			851
797	$x \equiv [x] \bullet$		852
798	$\text{let } x = M \text{ in } N \equiv \text{let } [x] = \Lambda \bullet. M \text{ in } N$		853
799			854
800	<b>D System F</b>		855
801	D.1 Syntax of System F		856
802			857
803	Type Variables	$a, b, c$	858
804	Type Constructors	$D ::= \text{Int} \mid \text{List} \mid \rightarrow \mid \times \mid \dots$	859
805	Types	$A, B ::= a \mid D \bar{A} \mid \forall a. A$	860
806	Type Contexts	$\Delta ::= \cdot \mid \Delta, a$	861
807	Term Contexts	$\Gamma ::= \cdot \mid \Gamma, x : A$	862
808	Term Variables	$x, y, z$	863
809	Terms	$M, N ::= [x] \mid \lambda(x : A). M \mid MN \mid \Lambda a. M \mid MA$	864
810			865
811	D.2 Type System of System F		866
812	Well-formed types.	$\boxed{\Delta \vdash A \text{ ok}}$	867
813			868
814		$\text{arity}(D) = n$	869
815	$a \in \Delta$	$\frac{\Delta \vdash A_1 \text{ ok} \dots \Delta \vdash A_n \text{ ok}}{\Delta \vdash D \bar{A} \text{ ok}}$	870
816			871
817	$\Delta \vdash a \text{ ok}$	$\frac{\Delta, a \vdash A \text{ ok}}{\Delta \vdash \forall a. A \text{ ok}}$	872
818	<b>Typing.</b>	$\boxed{\Delta; \Gamma \vdash M : A}$	873
819			874
820			875
821	APP		876
822	VAR	$\Delta; \Gamma \vdash M : A \rightarrow B$	877
823	$x : A \in \Gamma$	$\frac{\Delta; \Gamma \vdash N : A}{\Delta; \Gamma \vdash MN : B}$	878
824		$\frac{\Delta; \Gamma \vdash \lambda a. M : \forall a. B}{\Delta; \Gamma \vdash \lambda(x : A). M : A \rightarrow B}$	879
825	$\Delta; \Gamma \vdash [x] : A$	$\frac{\Delta; \Gamma \vdash M : \forall a. B}{\Delta; \Gamma \vdash MA : B[A/a]}$	880

881	<b>D.3 Equational Rules of System F</b>	936
882	As in Section 3 the equivalence $M \simeq N$ on terms $M$ and $N$ is defined only when $M$ and $N$ have the same type in the same context (i.e., $\Delta; \Gamma \vdash M : A$ and $\Delta; \Gamma \vdash N : A$ ).	937
883		938
884		939
885	$\beta$ -rules	940
886	$(\lambda(x : A).M)N \simeq M[N/\lceil x \rceil]$	941
887	$(\Lambda a.M)A \simeq M[A/a]$	942
888	$\eta$ -rules	943
889	$\lambda(x : A).M \lceil x \rceil \simeq M$	944
890	$\Lambda a.M a \simeq M$	945
891		946
892	<b>E Explicit FreezeML</b>	947
893	<b>E.1 Syntax of Explicit FreezeML</b>	948
894		949
895	<i>Types.</i>	950
896	Type Variables	951
897	Type Constructors	952
898	Types	953
899	Monotypes	954
900	Guarded Types	955
901	Monomorphic Instantiation	956
902	Polymorphic Instantiation	957
903	Type Contexts	958
904	Term Contexts	959
905		960
906	<i>Terms.</i>	961
907	I Term $\ni I, J ::= \lceil x \rceil$	962
908	$\lambda(x : A).I \mid I Q$	963
909	$\Lambda a.I \mid I A$	964
910	$\text{let } \lceil x \rceil = I \text{ in } J$	965
911		966
912	$\Lambda \bullet.P$	967
913		968
914	$M \bullet$	969
915		970
916	<b>E.2 Type System of Explicit FreezeML</b>	971
917		972
918	<i>Well-formed types.</i> $\boxed{\Delta \vdash A \text{ ok}}$	973
919	$a \in \Delta$	974
920	$\frac{\text{arity}(D) = n}{\Delta \vdash D \bar{A} \text{ ok}}$	975
921	$\Delta \vdash A_1 \text{ ok} \quad \dots \quad \Delta \vdash A_n \text{ ok}$	976
922	$\frac{\Delta, a \vdash A \text{ ok}}{\Delta \vdash \forall a.A \text{ ok}}$	977
923	<i>Monomorphic instantiation.</i> $\boxed{\Delta \vdash \sigma : \Delta' \Rightarrow_{\bullet} \Delta''}$	978
924		979
925	$\frac{}{\Delta \vdash \emptyset : \cdot \Rightarrow_{\bullet} \Delta'}$	980
926	$\frac{\Delta \vdash \sigma : \Delta' \Rightarrow_{\bullet} \Delta'' \quad \Delta, \Delta'' \vdash S \text{ ok}}{\Delta \vdash \sigma[a \mapsto S] : (\Delta', a) \Rightarrow_{\bullet} \Delta''}$	981
927		982
928	<i>Polymorphic instantiation.</i> $\boxed{\Delta \vdash \delta : \Delta' \Rightarrow_{\star} \Delta''}$	983
929		984
930		985
931	$\frac{}{\Delta \vdash \emptyset : \cdot \Rightarrow_{\star} \Delta'}$	986
932	$\frac{\Delta \vdash \delta : \Delta' \Rightarrow_{\star} \Delta'' \quad \Delta, \Delta'' \vdash A \text{ ok}}{\Delta \vdash \delta[a \mapsto A] : (\Delta', a) \Rightarrow_{\star} \Delta''}$	987
933		988
934		989
935		990

991	<i>Principality judgement.</i>	$\boxed{\text{principal}(\Delta, \Gamma, P, \Delta', A')}$	1046	
992			1047	
993			1048	
994			1049	
995			1050	
996		$\text{principal}(\Delta, \Gamma, P, \Delta', A') =$	1051	
997		$\Delta' = \text{ftv}(A') - \Delta$ and $\Delta, \Delta'; \Gamma \vdash P : A'$ and	1052	
998		(for all $\Delta'', A''$   if $\Delta'' = \text{ftv}(A'') - \Delta$ and	1053	
999		$\Delta, \Delta''; \Gamma \vdash P : A''$	1054	
1000		then there exists $\delta$ such that	1055	
1001		$\Delta \vdash \delta : \Delta' \Rightarrow_{\star} \Delta''$ and $\delta(A') = A''$ )	1056	
1002			1057	
1003	<i>Typing judgement.</i>	$\boxed{\Delta; \Gamma \vdash P : A}$	1058	
1004			1059	
1005			1060	
1006			1061	
1007			1062	
1008	VAR	LAM	APP	1063
1009	$x : A \in \Gamma$	$\Delta; \Gamma, x : A \vdash P : B$	$\Delta; \Gamma \vdash Q : A$	1064
1010		$\Delta; \Gamma \vdash \lambda(x : A).P : A \rightarrow B$	$\Delta; \Gamma \vdash P Q : B$	1065
1011			$\Delta, a; \Gamma \vdash I : B$	1066
1012			$\Delta; \Gamma \vdash \Lambda a.I : \forall a.B$	1067
1013		LET		1068
1014		$\Delta; \Gamma \vdash M : A \quad \Delta; \Gamma, x : A \vdash N : B$	$\Delta; \Gamma, x : S \vdash M : B$	1069
1015		$\Delta; \Gamma \vdash \text{let } [x] = M \text{ in } N : B$	$\Delta; \Gamma \vdash \lambda x.M : S \rightarrow B$	1070
1016				1071
1017	GEN		MONOINST	1072
1018	$\Delta, \Delta'; \Gamma \vdash M : A'$	$\text{principal}(\Delta, \Gamma, M, \Delta', A')$	$\Delta; \Gamma \vdash P : \forall \Delta'.G$	1073
1019			$\Delta \vdash \sigma : \Delta' \Rightarrow_{\star} \cdot$	1074
1020		$\Delta; \Gamma \vdash \Lambda \bullet. M : \forall \Delta'.A'$	$\Delta; \Gamma \vdash P \bullet : \sigma(G)$	1075
1021				1076
1022	<b>E.3 Equational Rules of Explicit FreezeML</b>			1077
1023	As in Section 3 the equivalence $P \simeq Q$ on terms $P$ and $Q$ is defined only when $P$ and $Q$ have the same type in the same context (i.e., $\Delta; \Gamma \vdash P : A$ and $\Delta; \Gamma \vdash Q : A$ ).			1078
1024				1079
1025				1080
1026				1081
1027				1082
1028		$\beta$ -rules	$(\lambda(x : A).P)Q \simeq P[Q/x]$	1083
1029			$(\Lambda a.I)A \simeq I[A/a]$	1084
1030				1085
1031		$\eta$ -rules	$\lambda(x : A).P[x] \simeq P$	1086
1032			$\Lambda a.I a \simeq I$	1087
1033				1088
1034		elaboration rules	$\text{let } [x] = M \text{ in } Q \simeq (\lambda(x : A).Q)M$	1089
1035			$\lambda x.P \simeq \lambda(x : S).P$	1090
1036			$\Lambda \bullet.I \simeq \Lambda \Delta.I$	1091
1037			$P \bullet \simeq P S_1 \dots S_n$	1092
1038			$P \star \simeq P A_1 \dots A_n$	1093
1039				1094
1040				1095
1041				1096
1042				1097
1043				1098
1044				1099
1045				1100

## 1101 E.4 Translation from Explicit FreezeML to System F

$$\begin{array}{lll}
 \text{1102} & \left[ \left[ \frac{x : A \in \Gamma}{\Delta ; \Gamma \vdash [x] : A} \right] = x \right. & \text{1156} \\
 \text{1103} & \left. \left[ \frac{\Delta ; \Gamma, x : A \vdash P : B}{\Delta ; \Gamma \vdash \lambda(x : A).P : A \rightarrow B} \right] = \lambda(x : A).\llbracket P \rrbracket \right. & \text{1157} \\
 \text{1104} & \left. \left[ \frac{\Delta ; \Gamma \vdash P : A \rightarrow B \quad \Delta ; \Gamma \vdash Q : A}{\Delta ; \Gamma \vdash PQ : B} \right] = \llbracket P \rrbracket \llbracket Q \rrbracket \right. & \text{1158} \\
 \text{1105} & & \text{1159} \\
 \text{1106} & & \text{1160} \\
 \text{1107} & \left[ \frac{\Delta, a ; \Gamma \vdash I : A}{\Delta ; \Gamma \vdash \Lambda a. I : \forall a. A} \right] = \Lambda a.\llbracket I \rrbracket & \text{1161} \\
 \text{1108} & & \text{1162} \\
 \text{1109} & & \left[ \frac{\Delta ; \Gamma \vdash P : \forall a. B}{\Delta ; \Gamma \vdash PA : B[A/a]} \right] = \llbracket P \rrbracket A & \text{1163} \\
 \text{1110} & & \text{1164} \\
 \text{1111} & \left[ \frac{\Delta ; \Gamma \vdash M : A \quad \Delta ; \Gamma, x : A \vdash Q : B}{\Delta ; \Gamma \vdash \text{let } [x] = M \text{ in } Q : B} \right] = (\lambda(x : A).\llbracket Q \rrbracket) \llbracket M \rrbracket & \text{1165} \\
 \text{1112} & & \left[ \frac{\Delta ; \Gamma, x : S \vdash P : B}{\Delta ; \Gamma \vdash \lambda x. P : S \rightarrow B} \right] = \lambda(x : S).\llbracket P \rrbracket & \text{1166} \\
 \text{1113} & & \text{1167} \\
 \text{1114} & & \text{1168} \\
 \text{1115} & \left[ \frac{\Delta, \Delta' ; \Gamma \vdash P : A' \quad \text{principal}(\Delta, \Gamma, P, \Delta', A')}{\Delta ; \Gamma \vdash \Lambda \bullet. P : \forall \Delta'. A'} \right] = \Lambda \Delta'. \llbracket P \rrbracket & \text{1169} \\
 \text{1116} & & \left[ \frac{\Delta ; \Gamma \vdash P : \forall \Delta'. G}{\Delta \vdash \sigma : \Delta' \Rightarrow_{\bullet} \cdot} \right] = \llbracket P \rrbracket \sigma(\Delta') & \text{1170} \\
 \text{1117} & & \text{1171} \\
 \text{1118} & & \text{1172} \\
 \text{1119} & & \text{1173} \\
 \text{1120} & & \left[ \frac{\Delta ; \Gamma \vdash P : \forall \Delta'. G}{\Delta \vdash \delta : \Delta' \Rightarrow_{\star} \cdot} \right] = \llbracket P \rrbracket \delta(\Delta') = & \text{1174} \\
 \text{1121} & & \text{1175} \\
 \text{1122} & & \left[ \frac{\Delta ; \Gamma \vdash P \star : \delta(G)}{\Delta ; \Gamma \vdash P \star : \delta(G)} \right] & \text{1176} \\
 \text{1123} & & \text{1177} \\
 \text{F} & \text{FreezeML} & \text{1178} \\
 \text{1124} & \text{F.1 Syntax of FreezeML} & \text{1179} \\
 \text{1125} & \text{Types.} & \text{1180} \\
 \text{1126} & & \text{1181} \\
 \text{1127} & \text{Type Variables} & \text{1182} \\
 \text{1128} & & \text{1183} \\
 \text{1129} & \text{Type Constructors} & \text{1184} \\
 \text{1130} & & \text{1185} \\
 \text{1131} & \text{Types} & \text{1186} \\
 \text{1132} & & \text{1187} \\
 \text{1133} & \text{Monotypes} & \text{1188} \\
 \text{1134} & & \text{1189} \\
 \text{1135} & \text{Guarded Types} & \text{1190} \\
 \text{1136} & & \text{1191} \\
 \text{1137} & \text{Polymorphic Instantiation} & \text{1192} \\
 \text{1138} & & \text{1193} \\
 \text{1139} & & \text{1194} \\
 \text{1140} & \text{Term Variables} & \text{1195} \\
 \text{1141} & & \text{1196} \\
 \text{1142} & & \text{1197} \\
 \text{1143} & \text{Type Contexts} & \text{1198} \\
 \text{1144} & \text{Term Contexts} & \text{1199} \\
 \text{1145} & & \text{1200} \\
 \text{1146} & & \text{1201} \\
 \text{1147} & & \text{1202} \\
 \text{1148} & \text{Terms} & \text{1203} \\
 \text{1149} & P, Q ::= x \mid [x] \mid \lambda x. P & \text{1204} \\
 \text{1150} & \mid \lambda(x : A). P \mid PQ & \text{1205} \\
 \text{1151} & \mid \text{let } x = P \text{ in } Q & \text{1206} \\
 \text{1152} & \mid \text{let } (x : A) = P \text{ in } Q & \text{1207} \\
 \text{1153} & \text{Well-formed types. } \boxed{\Delta \vdash A \text{ ok}} & \text{1208} \\
 \text{1154} & & \text{1209} \\
 \text{1155} & \frac{a \in \Delta}{\Delta \vdash a \text{ ok}} & \text{1210} \\
 \text{1156} & \frac{\text{arity}(D) = n \quad \Delta \vdash A_1 \text{ ok} \quad \dots \quad \Delta \vdash A_n \text{ ok}}{\Delta \vdash D \bar{A} \text{ ok}} & \\
 \text{1157} & \frac{\Delta, a \vdash A \text{ ok}}{\Delta \vdash \forall a. A \text{ ok}} & \\
 \text{1158} & \text{Polymorphic instantiation. } \boxed{\Delta \vdash \delta : \Delta' \Rightarrow_{\star} \Delta''} & \\
 \text{1159} & & \\
 \text{1160} & \frac{\Delta \vdash \emptyset : \cdot \Rightarrow_{\star} \Delta'}{\Delta \vdash \delta : \Delta' \Rightarrow_{\star} \Delta''} & \\
 \text{1161} & \frac{\Delta, \Delta'' \vdash A \text{ ok}}{\Delta \vdash \delta[a \mapsto A] : (\Delta', a) \Rightarrow_{\star} \Delta''} & \\
 \text{1162} & & \\
 \text{1163} & & \\
 \text{1164} & & \\
 \text{1165} & & \\
 \text{1166} & & \\
 \text{1167} & & \\
 \text{1168} & & \\
 \text{1169} & & \\
 \text{1170} & & \\
 \text{1171} & & \\
 \text{1172} & & \\
 \text{1173} & & \\
 \text{1174} & & \\
 \text{1175} & & \\
 \text{1176} & & \\
 \text{1177} & & \\
 \text{1178} & & \\
 \text{1179} & & \\
 \text{1180} & & \\
 \text{1181} & & \\
 \text{1182} & & \\
 \text{1183} & & \\
 \text{1184} & & \\
 \text{1185} & & \\
 \text{1186} & & \\
 \text{1187} & & \\
 \text{1188} & & \\
 \text{1189} & & \\
 \text{1190} & & \\
 \text{1191} & & \\
 \text{1192} & & \\
 \text{1193} & & \\
 \text{1194} & & \\
 \text{1195} & & \\
 \text{1196} & & \\
 \text{1197} & & \\
 \text{1198} & & \\
 \text{1199} & & \\
 \text{1200} & & \\
 \text{1201} & & \\
 \text{1202} & & \\
 \text{1203} & & \\
 \text{1204} & & \\
 \text{1205} & & \\
 \text{1206} & & \\
 \text{1207} & & \\
 \text{1208} & & \\
 \text{1209} & & \\
 \text{1210} & & \\
 \end{array}$$

1211 **Principality judgement.**  $\boxed{\text{principal}(\Delta, \Gamma, P, \Delta', A')}$  1266

1212

1213

1214

1215

1216

1217

1218

1219

$$\begin{aligned} \text{principal}(\Delta, \Gamma, P, \Delta', A') = \\ \Delta' = \text{ftv}(A') - \Delta \text{ and } \Delta, \Delta'; \Gamma \vdash P : A' \text{ and} \\ (\text{for all } \Delta'', A'' \mid \text{if } \Delta'' = \text{ftv}(A'') - \Delta \text{ and} \\ \Delta, \Delta''; \Gamma \vdash P : A'' \\ \text{then there exists } \delta \text{ such that} \\ \Delta \vdash \delta : \Delta' \Rightarrow_{\star} \Delta'' \text{ and } \delta(A') = A'') \end{aligned}$$

1220

1221

1222

1223

**Typing judgement.**  $\boxed{\Delta; \Gamma \vdash P : A}$

In contrast to Emrich et al. [4], we first present a simplified variant of FreezeML that does not incorporate the value restriction. In Appendix G we describe how to adapt the following to support the value restriction.

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

### F.3 Desugaring from FreezeML to Explicit FreezeML

1246

1247

1248

1249

1250

1251

1252

## G Incorporating the Value Restriction

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

$$\frac{\text{VAR}}{x : A \in \Gamma} \quad \frac{\text{VARINST}}{x : \forall \Delta'. G \in \Gamma} \quad \frac{\Delta \vdash \delta : \Delta' \Rightarrow_{\star} \cdot}{\Delta; \Gamma \vdash x : \delta(G)}$$

$$\frac{\text{U-LAM}}{\Delta; \Gamma, x : S \vdash P : B} \quad \frac{\text{LAM}}{\Delta; \Gamma, x : A \vdash P : B} \quad \frac{}{\Delta; \Gamma \vdash \lambda x. P : S \rightarrow B} \quad \frac{}{\Delta; \Gamma \vdash \lambda(x : A). P : A \rightarrow B}$$

$$\frac{\text{APP}}{\Delta; \Gamma \vdash P : A \rightarrow B \quad \Delta; \Gamma \vdash Q : A} \quad \frac{}{\Delta; \Gamma \vdash P Q : B}$$

$$\frac{\text{LETGEN}}{\Delta' = \text{ftv}(A') - \Delta \quad A = \forall \Delta'. A' \quad \Delta, \Delta''; \Gamma \vdash P : A' \quad \Delta; \Gamma, x : A \vdash Q : B} \quad \text{principal}(\Delta, \Gamma, P, \Delta', A')} \quad \frac{}{\Delta; \Gamma \vdash \text{let } x = P \text{ in } Q : B}$$

$$\frac{\text{A-LETGEN}}{A = \forall \Delta'. G \quad \Delta, \Delta'; \Gamma \vdash P : G \quad \Delta; \Gamma, x : A \vdash Q : B} \quad \frac{}{\Delta; \Gamma \vdash \text{let } (x : A) = P \text{ in } Q : B}$$

$$\begin{aligned} x &\equiv [x]_{\star} \\ \text{let } x = P \text{ in } Q &\equiv \text{let } [x] = \Lambda \bullet. P \text{ in } Q \\ (P : A) &\equiv (\lambda(x : A). [x]) P \\ (\Lambda \bullet. P : \forall \Delta. G) &\equiv \Lambda \Delta. (P : G) \\ \text{let } (x : A) = P \text{ in } Q &\equiv (\lambda(x : A). Q) (\Lambda \bullet. P : A) \end{aligned}$$

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

**Syntax.** We define the grammar of syntactic values as follows.

$$\text{Val} \ni V, W ::= x \mid \lambda x. M \mid \text{let } x = V \text{ in } W$$

1321 **Typing.** We define the following helper function. 1376

1322

$$1323 \text{gen}(\Delta, A, M) = \begin{cases} \text{ftv}(A) - \Delta & \text{if } M \in \text{Val} \\ . & \text{otherwise} \end{cases}$$

1324

1325 We then replace the ML typing rule LETGEN of the syntax-directed variant of ML (Appendix C.2) by the following rule. 1380

1326

1327

$$\frac{\text{LETGEN}}{\Delta' = \text{gen}(\Delta, S, M) \quad \Delta, \Delta'; \Gamma \vdash M : S \\ E = \forall \Delta'. S \quad \Delta; \Gamma, x : E \vdash N : T} \quad \Delta; \Gamma \vdash \text{let } x = M \text{ in } N : T$$

1328

1329 To adapt the orthogonal presentation, it suffices to limit the rule I-GEN-LAX to syntactic values. 1387

1330

1331 **Desugaring to Explicit ML.** We replace the desugaring rule for let with the following: 1388

1332

1333

$$\frac{\text{let } x = V \text{ in } N \equiv \text{let } [x] = \Lambda \bullet. V \text{ in } N}{\text{let } x = M \text{ in } N \equiv \text{let } [x] = M \text{ in } N} \quad \text{if } M \notin \text{Val}$$

1334

## G.2 FreezeML

1335 **Syntax.** The grammar is augmented as follows: 1395

1336

1337 Monomorphic Instantiation  $\sigma ::= \emptyset \mid \sigma[a \mapsto S]$

1338 Values  $\text{Val} \ni V, W ::= x \mid [x] \mid \lambda x. P \mid \lambda(x : A). P \mid \text{let } x = V \text{ in } W \mid \text{let } (x : A) = V \text{ in } W$

1339 Guarded Values  $\text{GVal} \ni U ::= x \mid \lambda x. P \mid \lambda(x : A). P \mid \text{let } x = V \text{ in } U \mid \text{let } (x : A) = V \text{ in } U$

1340 **Typing.** We define the following helper judgements and functions. 1401

1341

1342  $\boxed{\Delta \vdash \sigma : \Delta' \Rightarrow_{\bullet} \Delta''}$

1343  $\frac{}{\Delta \vdash \emptyset : \cdot \Rightarrow_{\bullet} \Delta'}$

1344  $\frac{\Delta \vdash \sigma : \Delta' \Rightarrow_{\bullet} \Delta'' \quad \Delta, \Delta'' \vdash S \text{ ok}}{\Delta \vdash \sigma[a \mapsto S] : (\Delta', a) \Rightarrow_{\bullet} \Delta''}$

1345  $\boxed{(\Delta, \Delta', P, A') \Downarrow A}$

1346  $\frac{P \in \text{GVal}}{(\Delta, \Delta', P, A') \Downarrow \forall \Delta'. A'}$

1347  $\frac{\Delta \vdash \sigma : \Delta' \Rightarrow_{\bullet} \cdot \quad P \notin \text{GVal}}{(\Delta, \Delta', P, A') \Downarrow \sigma(A')}$

1348  $\text{gen}(\Delta, A, P) = \begin{cases} (\Delta', \Delta') & \text{if } P \in \text{GVal} \\ (\cdot, \Delta') & \text{otherwise} \end{cases}$

1349 where  $\Delta' = \text{ftv}(A) - \Delta$

1350  $\text{split}(\forall \Delta. G, P) = \begin{cases} (\Delta, G) & \text{if } P \in \text{GVal} \\ (\cdot, \forall \Delta. G) & \text{otherwise} \end{cases}$

1351 (The judgement  $\Delta \vdash \sigma : \Delta' \Rightarrow_{\bullet} \Delta''$  is the monomorphic instantiation judgement of Explicit FreezeML.) 1419

1352 We replace the FreezeML typing rules LETGEN and A-LETGEN with the following rules. 1420

1353

$$\frac{\text{LETGEN}' \quad (\Delta', \Delta'') = \text{gen}(\Delta, A', P) \quad (\Delta, \Delta'', P, A') \Downarrow A \quad \Delta, \Delta''; \Gamma \vdash P : A' \quad \Delta; \Gamma, x : A \vdash Q : B \\ \text{principal}(\Delta, \Gamma, P, \Delta'', A')} {\Delta; \Gamma \vdash \text{let } x = P \text{ in } Q : B}$$

1354

1355  $\text{A-LETGEN}' \quad (\Delta', A') = \text{split}(A, P) \quad \Delta, \Delta'; \Gamma \vdash P : A' \quad \Delta; \Gamma, x : A \vdash Q : B$

1356  $\frac{}{\Delta; \Gamma \vdash \text{let } (x : A) = P \text{ in } Q : B}$

1431	<b>Desugaring to Explicit FreezeML.</b>	We replace the desugaring rule for <code>let</code> with the following two rules according to whether	1486
1432	the bound term is a guarded value or not.		1487
1433	<code>let x = U in Q</code>	$\equiv$	1488
1434	<code>let x = P in Q</code>	$\equiv$	1489
1435	<code>let [x] = <math>\Lambda \bullet. U</math> in Q</code>		1490
1436	<code>let [x] = <math>(\Lambda \bullet. \lambda(). P) \bullet ()</math> in Q</code>	if $P \notin \text{GVal}$	1491
1437	Here, <code>()</code> is the usual data constructor of the unit type and thunking enables us to treat $P$ as a value, as per the value restriction.		1492
1438	We replace the desugaring rule for type-annotated <code>let</code> with the following two rules.		1493
1439	<code>let (x : A) = U in Q</code>	$\equiv$	1494
1440	<code>let (x : A) = P in Q</code>	$\equiv$	1495
1441	$(\lambda(x : A). Q) (\Lambda \bullet. U : A)$		1496
1442	$(\lambda(x : A). Q) P$	if $P \notin \text{GVal}$	1497
1443	We rely on the syntactic sugar for type-annotated terms and type-annotated generalisation from Section 3; the latter being		1498
1444	restricted appropriately to accommodate the value restriction.		1499
1445	$(P : A) \equiv (\lambda(x : A). [x]) P$		1500
1446	$(\Lambda \bullet. U : \forall \Delta. G) \equiv \Lambda \Delta. (U : G)$		1501
1447			1502
1448			1503
1449			1504
1450			1505
1451			1506
1452			1507
1453			1508
1454			1509
1455			1510
1456			1511
1457			1512
1458			1513
1459			1514
1460			1515
1461			1516
1462			1517
1463			1518
1464			1519
1465			1520
1466			1521
1467			1522
1468			1523
1469			1524
1470			1525
1471			1526
1472			1527
1473			1528
1474			1529
1475			1530
1476			1531
1477			1532
1478			1533
1479			1534
1480			1535
1481			1536
1482			1537
1483			1538
1484			1539
1485			1540