The Verse Language: its type system and semantics

June 30, 2026

TIM SWEENEY, JAN VITEK, ANDY SONNENBERG, SIMON PEYTON JONES, LENNART AUGUSTSSON, KOEN CLAESSEN, STEPHANIE WEIRICH (EPIC GAMES)

ACM Reference Format:
Tim Sweeney, Jan Vitek, Andy Sonnenberg, Simon Peyton Jones, Lennart Augustsson, Koen Claessen, Stephanie Weirich (Epic Games). 2026. The Verse Language: its type system and semantics: June 30, 2026. 1, 1 (June 2026), 24 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 OVERVIEW

This document is a large collection of figures about the Verse language, covering various aspects including:

- Its syntax
- Rewrite semantics
- Verification semantics
- Denotational semantics

There is no accompanying text! Rather, these figures are reference materal for Simon's OPLSS talks. All this material is work in progress. It is not yet "settled", and there may well be errors or omissions. Please help us make them better.

2 SYNTAX

2.1 Source Verse

The syntax of Source Verse is given in Figure 1 and Figure 2.

2.2 Desugaring Source Verse into Essential Verse

Source Verse is desugared into Essential Verse using rewrite rules given in Figure 3 and Figure 4.

2.3 Essential Verse

The syntax of Essential Verse is in Figure 5.

Author's address: Tim Sweeney, Jan Vitek, Andy Sonnenberg, Simon Peyton Jones, Lennart Augustsson, Koen Claessen, Stephanie Weirich (Epic Games).

ACM XXXX-XXXX/2026/6-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Terms	t	::=	x	Variable
		|	$\_$	Wildcard
		|	k	Constant
		|	$t_1;\ t_2$	Sequencing
		|	$t_1[t_2]$	Function application
		|	$t_1(t_2)$	Non-failing function application
		|	$\exists x.\ t$	Existential
		|	$t_1 \mid t_2$	Choice
		|	$t_1 = t_2$	Unification
		|	**if** (t_1) **then** t_2 **else** t_3	Conditional expression
		|	**for** (t_1) **do** t_2	Finite loop
		|	**check**$\langle\omega\rangle\{t\}$	Effects check
		|	**array**$\{t_1;\cdots;\ t_n\}$	Array, $n \geqslant 0$
		|	$t_1, \cdots, t_n$	Array, $n \geqslant 2$
		|	$()$	Empty array
		|	$t_1 => t_2$	Lambda expression
		|	$p := t$	General definition
		|	$p : t$	Type ascription
		|	$op_b\ t$	Prefix operator
		|	$t\ op_a$	Postfix operator
		|	$t_1\ op_i\ t_2$	Infix operator
		|	$:t$	Range
		|	t_1 **where** t_2	Reverse sequencing
		|	**fun**$(t)\langle q\rangle\langle\omega\rangle\{t\}$	Function
		|	**if** (t) **then** t	Conditional expression
		|	**if** (t) **else** t	Conditional expression
		|	**if** $\{t\}$	Conditional expression
		|	**for** $\{t\}$	Finite loop
		|	**let** (t) **in** t	Local binding
		|	**block**$\{t\}$	Scope delimiter
		|	**case** (t) **of** $\{t_1;\cdots t_n\}$	Case expression
		|	**case** $\{t_1;\cdots t_n\}$	Case expression
		|	**type**$\{t\}$	Type definition
		|	**option**$\{\}$	Create an option
		|	**option**$\{t\}$	Create an option
		|	**truth**$\{t\}$	Singleton map
		|	**map**$\{t_1;\cdots;\ t_n\}$	Map, $n \geqslant 0$
Operators	op_b	::=	! | − | + | ^ | ? | [] | ..	Prefix
	op_a	::=	? | ^	Postfix
	op_i	::=	+ | − | ∗ | / | **and** | **or** | : | ..	Infix
		|	< | <= | > | >= | <> | −>	

Fig. 1. Source Verse expressions, part 1

Patterns∗	p	::=	$\_$	Wildcard
		|	xs	Variable
		|	$p_1 \to p_2$	Squiggle pattern
		|	$p(t)\ \langle\omega\rangle$	Function with effects
		|	$p(t)$	Short for $p(t)\ \langle$**succeeds**$\rangle$
		|	$p : t\ \langle\omega\rangle$	Type ascription
		|	$p : t$	Short for $p : t\ \langle$**decides**$\rangle$∗
		|	$:t$	Short for $(\_ : t)$
		|	$p_1, \cdots, p_n$	Tuple
		|	$..p$	Array splice
	xs	::=	$x \mid xs\,\&\,x$	
Effects	ω	::=	$\epsilon \mid \omega_1, \omega_2$	
		|	**fails** | **succeeds** | **decides**	
		|	...	
	$\omega_1 \cap \omega_2$	=	Effect intersection; e.g **succeeds** $\cap$ **decides** = **succeeds**	

Fig. 2. Source Verse expressions, part 2

Abbreviations for $p := e$ *

DSCOL1	$p : t = e$	$\rightarrow$	$p : t := e$	
DSCOL2	$:t = e$	$\rightarrow$	$:t := e$	
DSCOL3	$p : t$	$\rightarrow$	$p := :t$	

Rewriting $p := t$

DSWILD1	$\_ := t$	$\rightarrow$	t	
DSWILD2	$:t_1 := t_2$	$\rightarrow$	$(\_ : t_1) := t_2$	
DSPFUN1	$p(t_1)\ \langle oc, \omega\rangle := t_2$	$\rightarrow$	$p := \mathbf{fun}(t_1)\langle oc\rangle\langle\omega\rangle\{t_2\}$	
DSPFUN2	$p(t_1) := t_2$	$\rightarrow$	$p := \mathbf{fun}(t_1)\langle\mathbf{open}\rangle\langle\mathbf{succeeds}\rangle\{t_2\}$	
DSTY1	$p : t_1\ \langle\omega\rangle := t_2$	$\rightarrow$	$p := t_2 \rhd\langle\omega\rangle\ t_1$	
DSTY2*	$p : t_1 := t_2$	$\rightarrow$	$p : t_1\ \langle\mathbf{decides}\rangle := t_2$	
DSSQUIG	$(p_1 \rightarrow p_2) := t$	$\rightarrow$	$p_1 := x;\ p_2 := (x :\rightarrow t)$	x fresh
DSPARR	$\mathbf{ar}\{p_1; \cdots; p_n\} := t$	$\rightarrow$	$p_1 := x_1; \cdots; p_n := x_n$ $\mathbf{ar}\{x_1 : \mathbf{any}; \cdots; x_n : \mathbf{any}\} = t;$	none of p_i is $..p$
DSPAMP1	$xs\,\&\,x := t$	$\rightarrow$	$\mathbf{ar}\{xs\,\&\,x := t\}$	outside an array
DSPAMP2	$\mathbf{ar}\{\cdots; xs\,\&\,x := t; \cdots\}$	$\rightarrow$	$\mathbf{ar}\{\cdots; xs := t; x := t; \cdots\}$	inside an array

Application and unification

DSAPP	$t_1(t_2)$	$\rightarrow$	$f := t_1;\ a := t_2;\ \mathbf{check}\langle\mathbf{succeeds}\rangle\{f[a]\}$	

Array splice

DSSPLICE	$\mathbf{ar}\{t_1; \cdots; t_{k-1}; ..t_k; t_{k+1}; \cdots; t_n\}$	$\rightarrow$	$\mathbf{append}\,[\mathbf{ar}\{t_1; \cdots; t_{k-1}\}, \mathbf{append}[t_k, \mathbf{ar}\{t_{k+1}; \cdots; t_n\}]]$ where **append** is a reversible function appending two arrays	

Array notation

DSUNIT	$()$	$\rightarrow$	$\mathbf{ar}\{\}$	
DSTUPLE	$t_1, \cdots, t_n$	$\rightarrow$	$\mathbf{ar}\{t_1; \cdots; t_n\}$	

Map notation

DSMAP	$\mathbf{map}\{t_1, \cdots, t_n\}$	$\rightarrow$	$\mathbf{mkMap}(\mathbf{for}\ (f : \mathbf{ar}\{t_1; \cdots; t_n\};\ i \rightarrow a : f)\ \{(i, a)\})$	fresh f,i,a

Function notation

DSTYPE	$\mathbf{type}\{t\}$	$\rightarrow$	$\mathbf{fun}(x := t)\langle\mathbf{closed}\rangle\langle\mathbf{succeeds}\rangle\{x\}$	fresh x
DSFARROW	$t_1 => t_2$	$\rightarrow$	$\mathbf{fun}(t_1)\langle\mathbf{closed}\rangle\langle\mathbf{succeeds}\rangle\{t_2\}$	
DSFUN1	$\mathbf{fun}(a_1\ a_2 \cdots a_n)\langle oc\rangle\{t\}$	$\rightarrow$	$\mathbf{fun}(a_1)\langle oc\rangle\{\mathbf{fun}(a_2 \cdots a_n)\langle oc\rangle\{t\}\}$	
DSFUN2A	$\mathbf{fun}(a\langle\omega\rangle)\langle oc\rangle\{b\}$	$\rightarrow$	$\mathbf{fun}(a)\langle oc\rangle\langle\omega\rangle\{b\}$	
DSFUN2B	$\mathbf{fun}(a)\langle oc\rangle\{b\}$	$\rightarrow$	$\mathbf{fun}(a)\langle oc\rangle\langle\mathbf{succeeds}\rangle\{b\}$	

Conditional and for-loop notation

DSIF1	$\mathbf{if}\,\{e\}$	$\rightarrow$	$\mathbf{if}\,(e)\,\mathbf{else\ false}$	
DSIF2	$\mathbf{if}\,(e_1)\,\mathbf{then}\,e_2$	$\rightarrow$	$\mathbf{if}\,(e_1)\,\mathbf{then}\,e_2\,\mathbf{else\ false}$	
DSIF3	$\mathbf{if}\,(e_1)\,\mathbf{else}\,e_2$	$\rightarrow$	$\mathbf{if}\,(x := e_1)\,\mathbf{then}\,x\,\mathbf{else}\,e_2$	x fresh
DSFOR	$\mathbf{for}\,\{e\}$	$\rightarrow$	$\mathbf{for}\,(x := e)\,\{x\}$	x fresh

Fig. 3. Source Verse to Essential Verse rewriting (1)

Operators

DSPREQ	$?e$	$\rightarrow$	$\textbf{type}\,\{\textbf{false} \mid \textbf{let}\,(x:e)\,\{\textbf{truth}\{x\}\}\}$	x fresh
DSPOSTQ	$e?$	$\rightarrow$	$e[\_]$	
DSOP1	$op_b\ e$	$\rightarrow$	$\textbf{prefix}'\,op'_b\,[e]$	if op_b can fail
DSOP2	$op_b\ e$	$\rightarrow$	$\textbf{prefix}'\,op'_b\,(e)$	
DSOP3	$e\ op_a$	$\rightarrow$	$\textbf{postfix}'\,op'_a\,[e]$	if op_a can fail
DSOP4	$e\ op_a$	$\rightarrow$	$\textbf{postfix}'\,op'_a\,(e)$	
DSOP5	$e_1\ op_i\ e_2$	$\rightarrow$	$\textbf{operator}'\,op'_i\,[e_1, e_2]$	if op_i can fail
DSOP6	$e_1\ op_i\ e_2$	$\rightarrow$	$\textbf{operator}'\,op'_i\,(e_1, e_2)$	
DSAND	$e_1\ \textbf{and}\ e_2$	$\rightarrow$	$e_1;\ e_2$	
DSOR	$e_1\ \textbf{or}\ e_2$	$\rightarrow$	$\textbf{if}\,(e_1)\,\textbf{else if}\,(e_2)\,\textbf{else}:\textbf{false}$	
DSBANG	$!e$	$\rightarrow$	$\textbf{if}\ e\ \textbf{then}:\textbf{false}\ \textbf{else false}$	

Let, block, case

DSLET	$\textbf{let}\,(e)\,\textbf{in}\,b$	$\rightarrow$	$e;\ b$	*see note*
DSBLOCK	$\textbf{block}\{b\}$	$\rightarrow$	b	*see note*
DSCASE1	$\textbf{case}\,\{e_1;\cdots e_n\}$	$\rightarrow$	$(x:\textbf{any}) \Rightarrow \textbf{case}\,(x)\,\{e_1;\cdots e_n\}$	x fresh
DSCASE2	$\textbf{case}\,(e)\,\{e_1;\cdots e_n\}$	$\rightarrow$	$x := e;\ e_1[x]\ \textbf{or}\cdots\textbf{or}\ e_n[x]$	x fresh

[LA: This desugaring does not work for multivalued e_i.]

Misc

DSOPT1	$\textbf{option}\{\,\}$	$\rightarrow$	$\textbf{false}$	
DSOPT2	$\textbf{option}\{e\}$	$\rightarrow$	$\textbf{if}\,(x := e)\ \textbf{then truth}\{x\}\ \textbf{else false}$	x fresh

Note: The desugaring assumes a scope correct Verse program. This means that there can be no variable name clashes in **let** and **block**$\{\cdot\}$. A more robust desugaring could α-convert such clashes.

Fig. 4. Source Verse to Source Verse rewriting (2)

Variables	x, y, f			
Literals	ki	::=	*Integer literals*	
	kc	::=	*Character literals*	
	op	::=	*Primitive operations*	
	k	::=	$op \mid ki \mid kc \mid \cdots$	
Terms	t	::=	x	Variable
		$\mid$	$\_$	Underscore
		$\mid$	k	Constant
		$\mid$	$t_1[t_2]$	Function application
		$\mid$	$t_1; t_2$	Sequencing
		$\mid$	t_1 **where** t_2	Reverse sequencing
		$\mid$	$t_1 = t_2$	Unification
		$\mid$	$x := t$	Definition; binds x
		$\mid$	$x :\rightarrow t$	Capture input; binds x
		$\mid$	**block**$\{t\}$	Block; limits scope
		$\mid$	$t_1 \mid t_2$	Choice
		$\mid$	**if** (t_1) **then** t_2 **else** t_3	Conditional expression
		$\mid$	**for**(t_1) **do**$\{t_2\}$	Finite loop
		$\mid$	**array**$\{t_1, \cdots, t_n\}$	Array, $n \geqslant 0$
		$\mid$	**truth**$\{t\}$	Truth value
		$\mid$	**fun**$(t_1)\langle q\rangle\langle\omega\rangle\{t_2\}$	Function
		$\mid$	$:t$	Range
		$\mid$	$t_1 \rhd\langle\omega\rangle\ t_2$	Opacity
		$\mid$	**check**$\langle\omega\rangle\{t\}$	Effects check
Primops	o	::=	**gt** $\mid$ **add** $\mid$ **int**	
Aperture	q	::=	**open** $\mid$ **closed**	Open vs closed functions

Variables can be alphanumeric, but also **prefix**$'op'$, **postfix**$'op'$, and **operator**$'op'$.

Fig. 5. Essential Verse

Variables	x, y, z, f, g			
Literals	ki	$::=$	*Integer literals*	
	kc	$::=$	*Character literals*	
	op	$::=$	*Primitive operations*	
	k	$::=$	$op \mid ki \mid kc \mid \cdots$	
Prim operators	op	$::=$	$\mathbf{add} \mid \mathbf{sub} \mid \mathbf{gt} \mid \mathbf{isInt} \mid \cdots$	
Values	v	$::=$	$x \mid hnf$	
Head values	hnf	$::=$	$k \mid r \mid \langle v_1, \cdots, v_n \rangle \mid \lambda x.\, b \mid \mathbf{delay}\{e\}$	
Expressions	e	$::=$	$v \mid e_1[e_2]$	
		$\mid$	$e_1;\ e_2 \mid e_1 = e_2 \mid b_1 \,\mathbf{|}\, b_2 \mid \mathbf{fail}$	
		$\mid$	$\mathbf{iter}(ic)\{b\}\{e\}$	
		$\mid$	$x \rightsquigarrow_w t \mid \mathbf{block}\{b\}$	
		$\mid$	$\mathbf{verify}(R;\ A)\{b\}$	
		$\mid$	$e_1 \rhd\langle\omega\rangle\ e_2$	
Blocks	b	$::=$	$\exists X\{hp\}.\, e$	
Variable sets	X	$::=$	$\{x_1, \cdots, x_n\}$	$(n \geqslant 0)$
Heaps	hp	$::=$	$x_1 \leftarrow v_1 \ldots, x_n \leftarrow v_n$	$(n \geqslant 0)$
	w	$::=$	(bb, dr, vx)	
	bb	$::=$	$\circ \mid \bullet$	
	dr	$::=$	$D \mid R\langle\omega\rangle$	
	vx	$::=$	$\mathbf{v} \mid \mathbf{x}$	
Iter combinators	ic	$::=$	$\mathbf{all} \mid \mathbf{if} \mid \mathbf{for}$	
Effects	ω	$::=$	$\mathbf{succeeds} \mid \mathbf{decides} \mid \mathbf{fails}$	
Skolems	r, s, t			
Skolem sets	R	$::=$	$\{r_1, \cdots, r_n\}$	$(n \geqslant 0)$
Ground values	gv	$::=$	$k \mid r \mid \langle gv_1, \cdots, gv_n \rangle$	
Assumptions	a	$::=$	$r = gv \mid r \neq gv \mid r = op[gv] \mid \neg op[gv]$	
	A	$::=$	$\{a_1, \cdots, a_n\}$	$(n \geqslant 0)$
Evaluation contexts	C	$::=$	$\square \mid e = C \mid C = e \mid e;\ C \mid C;\ e \mid e[C] \mid C[e]$	
Substitution contexts	S	$::=$	$V \mid e = S \mid S = e \mid e;\ S \mid S;\ e \mid e[S] \mid S[e]$	
		$\mid$	$B[S] \,\mathbf{|}\, b \mid b \,\mathbf{|}\, B[S]$	
		$\mid$	$\mathbf{iter}(ic)\{B[S]\}\{e\} \mid \mathbf{iter}(ic)\{b\}\{S\}$	
Value contexts	V	$::=$	$\square \mid \langle v_1, \cdots, V, \cdots v_n \rangle \mid \lambda x.\, B[S]$	
Block contexts	B	$::=$	$\exists X\{hp\}.\, \square$	

Fig. 6. Core Verse Syntax

3 REWRITING

This section gives the operational semantics of Essential Verse using rewrite rules.

We rewrite terms written in the expression syntax of Figure 6, which includes terms of Essential Verse.

3.1 Auxiliary functions

$\mathsf{tbs}(t)$ Variables bound by Essential Verse term t
$e[v/x]$ Capture-avoiding substitution of v for x in e
$\mathsf{fvs}(e)$ Free variables of expression e
$\mathsf{bvs}(S)$ Bound variables of a context S
$\mathsf{fvsol}(e)$ Free variables outside lambdas in value v

$$
\begin{array}{rcl}
\mathsf{leftChoiceFree}(C) & :: & \mathit{Bool} \\
\mathsf{leftChoiceFree}(\square) & = & \mathit{True} \\
\mathsf{leftChoiceFree}(C = e) & = & \mathsf{leftChoiceFree}(C) \\
\mathsf{leftChoiceFree}(C;\ e) & = & \mathsf{leftChoiceFree}(C) \\
\mathsf{leftChoiceFree}(e = C) & = & \mathsf{choiceFree}(e) \wedge \mathsf{leftChoiceFree}(C) \\
\mathsf{leftChoiceFree}(e;\ C) & = & \mathsf{choiceFree}(e) \wedge \mathsf{leftChoiceFree}(C) \\
\mathsf{leftChoiceFree}(e[C]) & = & \mathit{False}
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{choiceFree}(e) & :: & \mathit{Bool} \\
\mathsf{choiceFree}(v) & = & \mathit{True} \\
\mathsf{choiceFree}(e_1;\ e_2) & = & \mathsf{choiceFree}(e_1) \wedge \mathsf{choiceFree}(e_2) \\
\mathsf{choiceFree}(e_1 = e_2) & = & \mathsf{choiceFree}(e_1) \wedge \mathsf{choiceFree}(e_2) \\
\mathsf{choiceFree}(e_1[e_2]) & = & \mathsf{choiceFree}(e_1) \wedge \mathsf{choiceFree}(e_2) \\
\mathsf{choiceFree}(\mathbf{fail}) & = & \mathit{True} \\
\mathsf{choiceFree}(b_1 \mid b_2) & = & \mathit{False} \\
\mathsf{choiceFree}(\mathbf{block}\{\exists X\{hp\}.\ e\}) & = & \mathsf{choiceFree}(e) \\
\mathsf{choiceFree}(\mathbf{iter}(\mathbf{all})\{b\}\{e\}) & = & \mathit{True} \\
\mathsf{choiceFree}(\mathbf{iter}(\mathbf{if})\{b\}\{e\}) & = & \mathit{False} \\
\mathsf{choiceFree}(\mathbf{iter}(\mathbf{for})\{b\}\{e\}) & = & \mathsf{choiceFree}(e)
\end{array}
$$

Fig. 7. Auxiliary functions

4 REWRITE RULES

This section give the rewrite rules. Some general observations.

- Many rules have a side condition like $x \notin \mathsf{fvs}(e)$. The idea is that we silently use the alpha-renaming rules (Figure 8) to make these side conditions hold.
- All rules are careful to keep non-value sub-expressions in the same order. This only matters in the future when effects (including I/O) play a bigger role. Example: the normalisation rules in Figure 8.

4.1 Substitution and promotion

$$
\begin{array}{lrcll}
\textsc{subst} & \exists X\{hp, x \leftarrow v\}.\ S[x] & \longrightarrow & \exists X\{hp, x \leftarrow v\}.\ S[v] & \text{if } x \notin \mathsf{bvs}(S) \\
\textsc{gc} & \exists X, Y\{hp\}.\ e & \longrightarrow & \exists X\{hp \setminus\setminus Y\}.\ e & \text{if } Y \# \mathsf{fvs}(\mathbf{rng}(hp \setminus\setminus Y), e),\ Y \neq \{\} \\
\textsc{promote-l} & \exists X, x\{hp\}.\ C[x = v] & \longrightarrow & \exists X, x\{hp[v/x], x \leftarrow v\}.\ C[v] & \\
 & & & & \text{if } x \notin \mathbf{dom}(hp),\ \mathsf{fvsol}(v) \# \mathbf{dom}(hp) \cup \{x\} \\
\textsc{promote-r} & \exists X, x\{hp\}.\ C[v = x] & \longrightarrow & \text{...just like } \textsc{promotel} &
\end{array}
$$

Fig. 8. Structural and normalisation rules

- Given a heap binding $x \leftarrow v$, rule SUBST substitutes v for a *single* occurrence of x. The context S (see Figure 6) allows us to look deep in the term, including under choices and lambdas.
- The GC rule allows us to garbage-collect a strongly-connected component of possibly-recursive bindings in the heap.

- You might wonder if we could eliminate PROMOTE-R in favour of a SWAP rule that swaps $x = hnf$ to $hnf = x$. But we must also deal with $x = y$ where y is bound locally and x is bound further out. We could have a swap rule that swaps $v = x$ but that could loop infinitely on $x = y$ which seems unpleasant.

4.2 Unification

EQK1	$k = k$	$\longrightarrow$	k	
EQK2	$k_1 = k_2$	$\longrightarrow$	**fail**	if $k_1 \neq k_2$
EQTUP1	$\langle v_1, \cdots, v_n \rangle = \langle w_1, \cdots, w_n \rangle$	$\longrightarrow$	$v_1 = w_1; \cdots v_n = w_n; \langle v_1, \cdots, v_n \rangle$	
EQTUP2	$\langle v_1, \cdots, v_n \rangle = \langle w_1, \cdots, w_m \rangle$	$\longrightarrow$	**fail**	if $n \neq m$
EQFAIL	$hnf_1 = hnf_2$	$\longrightarrow$	**fail**	if hnf_1 and hnf_2 have different root shapes

Fig. 9. Unification rules

- There is no rule for $(\lambda x.\, e_1) = (\lambda y.\, e_2)$ because the runtime can't test for equality of functions.
- Instead that equality is *stuck*. A stuck expression is our version of "goes wrong at runtime". It should never happen; the verifier should reject such a program before we run it.
- So (slightly surprisingly) there no rule for $x = x$ either, in case x is bound to a lambda. That equality is stuck until x gets a value.

4.3 Primitive operations

We just give a small representative selection here.

ADD	$\mathbf{add}[ki_1, ki_2]$	$\longrightarrow$	$ki_1 + ki_2$	
GT1	$\mathbf{gt}[ki_1, ki_2]$	$\longrightarrow$	ki_1	if $ki_1 > ki_2$
GT2	$\mathbf{gt}[ki_1, ki_2]$	$\longrightarrow$	**fail**	if $ki_1 \leqslant ki_2$
ISINT1	$\mathbf{isInt}[ki]$	$\longrightarrow$	ki	(ki is an integer)
ISINT2	$\mathbf{isInt}[v]$	$\longrightarrow$	**fail**	if v is not an integer
CONS	$\mathbf{cons}[v_0, \langle v_1, \cdots, v_n \rangle]$	$\longrightarrow$	$\langle v_0, v_1, \cdots, v_n \rangle$	
DOTDOT	$\mathbf{dotdot}[ki_1, ki_2]$	$\longrightarrow$	$ki_1 \mid (ki_1 + 1) \mid \cdots \mid ki_2$	

Fig. 10. Primitive operation rules

- An ill-typed application, such as $\mathbf{add}[3, \lambda x.\, x]$ where the argument(s) are not integers, is just stuck. Similarly for other primitive operators.

4.4 Beta reduction

BETA-L	$(\lambda x.\, b)[e]$	$\longrightarrow$	$\mathbf{block}\{\exists x\{\}.\, x = e;\, \mathbf{block}\{b\}\}$	if $x \notin \mathsf{fvs}(e)$
BETA-T	$\langle v_0, \cdots, v_n \rangle[e]$	$\longrightarrow$	$\mathbf{block}\{\exists x\{\}.\, x = e;\, ((\exists\{\}.\, x = 0;\, v_0) \mid \cdots \mid (\exists\{\}.\, x = n;\, v_n))\}$	

Fig. 11. Primitive operation rules

- Note that BETA-T does narrowing.
- In BETA-T the binding $x = e$ is there to avoid duplicating e.
- The base case of BETA-T for the empty tuple simply yields failure.

4.5 Structural and normalisation rules

$$
\begin{array}{lrcll}
\textsc{hp-alpha} & \exists X, x\{hp\}.\, e & \leftrightarrow & \exists X, y\{hp[y/x]\}.\, e[y/x] & \text{if } y \notin \mathsf{fvs}(hp, e) \\
\textsc{lam-alpha} & \lambda x.\, e & \leftrightarrow & \lambda y.\, e[x/y] & \text{if } y \notin \mathsf{fvs}(e) \\
 & & & & \\
\textsc{seq} & v;\, e & \longrightarrow & e & \\
\textsc{norm1} & v = (e_1;\, e_2) & \longrightarrow & e_1;\, v = e_2 & \\
\textsc{norm2} & v = (e_1 = e_2) & \longrightarrow & v = e_1;\, v = e_2 & \\
\textsc{norm3} & (e_1;\, e_2) = e_3 & \longrightarrow & e_1;\, e_2 = e_3 & \\
\textsc{norm4} & (v = e_1) = e_2 & \longrightarrow & v = e_1;\, v = e_2 & \\
\textsc{norm5} & (e_1;\, e_2);\, e_3 & \longrightarrow & e_1;\, (e_2;\, e_3) & \\
 & & & & \\
\textsc{float} & \exists X\{hx\}.\, C[\mathbf{block}\{\exists Y\{hy\}.\, e\}] & \longrightarrow & \exists X, Y\{hx, hy\}.\, C[e] & \\
 & & & \text{if } X \# Y,\, Y \notin \mathsf{fvs}(C),\, \mathsf{fvsol}(\mathbf{rng}(hy)) \# X &
\end{array}
$$

Fig. 12. Structural and normalisation rules

- NORM4 is necessary to make progress with $(x = p + 1) = 5$. We want to unify x with 5 which might in turn unlock unification for p.
- NORM5 is not strictly necessary, but it helps to de-clutter; e.g. $(e_1;\, 5);\, e_2$ rewrites to $e_1;\, e_2$.

4.6 Iteration and choice

$$
\begin{array}{lrcl}
\textsc{fail} & \mathbf{iter}(ic)\{B[C[\mathbf{fail}]]\}\{e\} & \longrightarrow & e \\
\textsc{choice} & \mathbf{iter}(ic)\{B[C[b_1 \mid b_2]]\}\{e\} & \longrightarrow & \mathbf{iter}\ (ic)\{B[C[\mathbf{block}\{b_1\}]]\} \\
 & & & \quad \{\mathbf{iter}(ic)\{B[C[\mathbf{block}\{b_2\}]]\}\{e3\}\} \\
 & & & \quad \text{if } \mathsf{leftChoiceFree}(C) \\
 & & & \\
\textsc{if} & \mathbf{iter}(\mathbf{if})\{B[\mathbf{delay}\{e_1\}]\}\{e_2\} & \longrightarrow & \mathbf{block}\{B[e_1]\} \\
\textsc{all} & \mathbf{iter}(\mathbf{all})\{B[v]\}\{e\} & \longrightarrow & \mathbf{cons}[B[v], e] \\
\textsc{for} & \mathbf{iter}(\mathbf{for})\{B[v]\}\{e\} & \longrightarrow & \mathbf{block}\{\exists x\{\}.\, cons2[\langle x, B[v]\rangle, e]\}
\end{array}
$$

Fig. 13. Iteration and choice

4.7 Matching

$$
\begin{array}{lrcl}
\textsc{mwild} & u \rightsquigarrow_w \_ & \longrightarrow & u \\
\textsc{mlit} & u \rightsquigarrow_w k & \longrightarrow & u = k \\
\textsc{mvar} & u \rightsquigarrow_w y & \longrightarrow & u = y \\
\textsc{mdef} & u \rightsquigarrow_w x := t & \longrightarrow & x = (u \rightsquigarrow_w t) \\
\textsc{minp} & u \rightsquigarrow_w x :\to t & \longrightarrow & x = u;\ u \rightsquigarrow_w t \\
\textsc{mblock} & u \rightsquigarrow_w \mathbf{block}\{t\} & \longrightarrow & \mathbf{block}\{\exists \mathsf{tbs}(t)\{\}.\ u \rightsquigarrow_w t\} \\
\textsc{mapp} & u \rightsquigarrow_w t_1[t_2] & \longrightarrow & u = \mathcal{D}_w[\![t_1]\!][\mathcal{D}_w[\![t_2]\!]] \\
\textsc{munif} & u \rightsquigarrow_w t_1 = t_2 & \longrightarrow & (u \rightsquigarrow_w t_1) = (u \rightsquigarrow_w t_2) \\
\textsc{mchoice} & u \rightsquigarrow_w t_1 \mid t_2 & \longrightarrow & (u \rightsquigarrow_w \mathbf{block}\{t_1\}) \mid (u \rightsquigarrow_w \mathbf{block}\{t_2\}) \\
\textsc{mfail} & u \rightsquigarrow_w \mathbf{fail} & \longrightarrow & \mathbf{fail} \\
\textsc{msemi} & u \rightsquigarrow_w t_1;\ t_2 & \longrightarrow & \mathcal{D}_w[\![t_1]\!];\ u \rightsquigarrow_w t_2 \\
\textsc{mwhere} & u \rightsquigarrow_w t_1\ \mathbf{where}\ t_2 & \longrightarrow & \exists y\{\}.\ y = (u \rightsquigarrow_w t_1);\ \mathcal{D}_w[\![t_2]\!];\ y \\
\textsc{mtup} & u \rightsquigarrow_w \mathbf{array}\{t_1, \cdots, t_n\} & \longrightarrow & \mathbf{block}\{\exists\ u_1 \cdots u_n, y_1 \cdots y_n\{\}.\ u = \langle u_1, \cdots, u_n\rangle; \\
 & & & \qquad y_1 = (u_1 \rightsquigarrow_w t_1);\ \cdots;\ y_n = (u_n \rightsquigarrow_w t_n); \\
 & & & \qquad \langle y_1, \cdots, y_n\rangle\} \\
\textsc{mcolon1} & x \rightsquigarrow_w :t & \longrightarrow & \mathcal{D}_w[\![t]\!][x] \\
\textsc{mif} & u \rightsquigarrow_w \mathbf{if}(t_0)\{t_1\}\mathbf{else}\{t_2\} & \longrightarrow & \mathbf{iter}\ (\mathbf{if})\{\exists \mathsf{tbs}(t_0)\{\}.\ \mathcal{D}_w[\![t_0]\!];\ \mathbf{delay}\{u \rightsquigarrow_w \mathbf{block}\{t_1\}\}\} \\
 & & & \qquad \{u \rightsquigarrow_w \mathbf{block}\{t_2\}\} \\
\textsc{mall} & u \rightsquigarrow_w \mathbf{for}\{t\} & \longrightarrow & u = \mathbf{iter}(\mathbf{all})\{\exists \mathsf{tbs}(t)\{\}.\ \mathcal{D}_w[\![t]\!]\}\{\langle\rangle\} \\
\textsc{mfor} & u \rightsquigarrow_w \mathbf{for}(t_0)\,\mathbf{do}\{t_1\} & \longrightarrow & \mathbf{block}\{\exists y\{\}. \\
 & & & \qquad \langle u, y\rangle = \mathbf{iter}\ (\mathbf{for})\{\exists \mathsf{tbs}(t_0)\{\}.\ \mathcal{D}_w[\![t_0]\!];\ \lambda u'.\ u' \rightsquigarrow_w \mathbf{block}\{t_1\}\} \\
 & & & \qquad\qquad \{\langle\langle\rangle, \langle\rangle\rangle\}; \\
 & & & \qquad y\}
\end{array}
$$

where $\mathcal{D}_{(\_,\_,v)}[\![t]\!] = \exists u\{\}.\ u \rightsquigarrow_{(\circ,D,v)} t$

Fig. 14. Matching: most cases

- In MWHERE we are careful to keep t_1 and t_2 in the same order, in case they have effects.

4.8 Matching functions

$$
\begin{array}{lrcl}
\textsc{mfun} & h \rightsquigarrow_w \mathbf{fun}(at)\{bt\} & \longrightarrow & \lambda u.\ \exists p, q, \mathsf{tbs}(at)\{\}.\ p = (u \rightsquigarrow_w at); \\
 & & & \qquad \mathcal{W}_{bb}[\![q = h[p]]\!]; \\
 & & & \qquad q \rightsquigarrow_w \mathbf{block}\{bt\}
\end{array}
$$

where

$$
\begin{array}{rcl}
(bb, dr, vx) & = & w \\
\mathcal{W}_\circ[\![e]\!] & = & \langle\rangle \\
\mathcal{W}_\bullet[\![e]\!] & = & e
\end{array}
$$

Fig. 15. Matching functions

4.9 Verification rules

$$
\begin{array}{lll}
\textsc{mfun} & h \rightsquigarrow_w \mathbf{fun}(at)\langle\omega\rangle\{bt\} & \longrightarrow \quad \mathbf{verify}(u;)\{\exists p, q, \mathsf{tbs}(at)\{\}.\ p = (u \rightsquigarrow_{w_1} at);\ \mathcal{W}_{bb}[\![q = h[p]]\!];\ \mathbf{check}\langle\omega\rangle\{q \rightsquigarrow_{w_2} \mathbf{block}\{bt\}\}\} \\
 & & \quad\ \lambda u.\ \exists p, q, \mathsf{tbs}(at)\{\}.\ p = (u \rightsquigarrow_{w_3} at);\ \mathcal{W}_{bb}[\![q = h[p]]\!];\ q \rightsquigarrow_{w_4} \mathbf{block}\{bt\}
\end{array}
$$

where

$$
\begin{array}{rcl}
(bb, \_, vx) & = & w \\
w_1 & = & (\circ, D, \mathbf{v}) \\
w_2 & = & (bb, R\langle\omega\rangle, \mathbf{v}) \\
w_3 & = & (\bullet, D, vx) \\
w_4 & = & (\bullet, R\langle\omega\rangle, \mathbf{x}) \\
\mathcal{W}_\circ[\![e]\!] & = & \langle\rangle \\
\mathcal{W}_\bullet[\![e]\!] & = & e
\end{array}
$$

$$
\begin{array}{lll}
\textsc{moftype} & u \rightsquigarrow_w t_1 \rhd t_2 & \longrightarrow\ u = \mathcal{D}_w[\![t_1]\!][\mathcal{D}_w[\![t_2]\!]] \qquad \text{if } w = (b, \mathbf{x}, D) \\
 & & \longrightarrow\ (u \rightsquigarrow_w t_1) \rhd\langle\omega\rangle\ \mathcal{D}_w[\![t_2]\!] \qquad \text{if } w = (b, \mathbf{x}, R\langle\omega\rangle) \\
 & & \longrightarrow\ \mathbf{block}\{\exists y, t\{\}.\ y = \mathcal{D}_w[\![t_1]\!];\ t = \mathcal{D}_w[\![t_2]\!];\ \mathbf{check}\langle\mathbf{succeeds}\rangle\{t[y]\};\ y \rhd t\ \} \qquad \text{if } w = (b, \mathbf{v}, D) \\
 & & \longrightarrow\ \mathbf{block}\{\exists y, t\{\}.\ y = \mathcal{D}_w[\![t_1]\!];\ t = \mathcal{D}_w[\![t_2]\!];\ \mathbf{check}\langle\mathbf{succeeds}\rangle\{t[y]\}\ t[y]\ \} \qquad \text{if } w = (b, \mathbf{v}, R\langle\_\rangle) \\
\textsc{oftype} & e_1 \rhd\langle\omega\rangle\ e_2 & \longrightarrow\ e_2[e_1] \qquad \text{Outside } \mathbf{verify}
\end{array}
$$

Fig. 16. Matching functions with verification

Proof contexts $P ::= \exists X\{hp\}.\, Q$

$Q ::= \square \mid e = Q \mid Q = e \mid e;\, Q \mid Q;\, e \mid e[Q] \mid Q[e] \mid \mathbf{iter}(ic)\{Q\}\{e\}$

SOLVER	$\mathbf{verify}(R;\, A)\{b\}$	$\longrightarrow$	$\langle\rangle$ if $R;\, A \vDash \mathit{inconsistent}$
V-VAL	$\mathbf{verify}(R;\, A)\{B[v]\}$	$\longrightarrow$	$\langle\rangle$
V-CHOICE	$\mathbf{verify}(R;\, A)\{P[e_1 \mid e_2]\}$	$\longrightarrow$	$\mathbf{verify}(R;\, A)\{P[e_1]\};\ \mathbf{verify}(R;\, A)\{P[e_2]\}$
V-FAIL	$\mathbf{verify}(R;\, A)\{P[\mathbf{fail}]\}$	$\longrightarrow$	$\langle\rangle$
SPLIT-VAL	$\mathbf{verify}(R;\, A)\{P[r = gv;\, e]\}$	$\longrightarrow$	$\mathbf{verify}(R;\, A, r = gv)\{P[e]\};$ $\mathbf{verify}(R;\, A, r \neq gv)\{P[\mathbf{fail}]\}$
SPLIT-OP	$\mathbf{verify}(R;\, A)\{P[op[gv]]\}$	$\longrightarrow$	$\mathbf{verify}(R, r;\, A, r = op[gv])\{P[r]\};$ $\mathbf{verify}(R;\, A, \neg op[gv])\{P[\mathbf{fail}]\}$ if $r \notin R$
SPLIT-TUP	$\mathbf{verify}(R;\, A)\{P[r = \langle v_1, \cdots, v_n\rangle;\, e]\}$	$\longrightarrow$	$\mathbf{verify}(R, r_1 \cdots r_n;\, A, r = \langle r_1, \cdots, r_n\rangle)$ $\{P[r_1 = v_1;\, \cdots;\, r_n = v_n;\, e]\};$ $\mathbf{verify}(R;\, A, r \neq \langle \_, \cdots, \_\rangle)\{P[\mathbf{fail}]\}$
SPLIT-APP	$\mathbf{verify}(R;\, A)\{P[f[gv]]\}$	$\longrightarrow$	$\mathbf{verify}(R, r;\, A, r = AP[f, gv])\{P[r]\};$ if $f \in R, r \notin R$
SPLIT-APPF	$\mathbf{verify}(R;\, A)\{P[f[\lambda x.\, e]]\}$	$\longrightarrow$	$\mathbf{verify}(R, r;\, A)\{P[r]\};$ if $f \in R, r \notin R$
V-SOME	$\mathbf{verify}(R;\, A)\{P[e_1 \rhd\langle\omega\rangle\ e_2]\}$	$\longrightarrow$	$\mathbf{verify}(R, r;\, A)\{\exists x.\, x = e_2[r];\, P[x]\};$ if ω can s $\mathbf{verify}(R;\, A)\{\exists x.\, x = e_2[r];\, P[\mathbf{fail}]\};$ if ω can f if $r \notin R$ and $\mathsf{fvs}(v) \,\#\, \mathsf{bvs}(P)$

Fig. 17. Verfication rules

U-OLAM	$\mathbf{olam}(z_1, v_1) = \mathbf{olam}(z_2, v_2);\ e$	$\longrightarrow$	$\exists z_3.\, z_1 = \mathbf{olam}(z_3, v_2);\ z_2 = \mathbf{olam}(z_3, v_1);\ e$
APP-OLAM	$(\mathbf{olam}(z, v_1))[v_2]$	$\longrightarrow$	$\exists x.\, x = z[v_2];\ x = v_1[v_2];\ x$
EXI-APP	$\exists z.\, e$	$\longrightarrow$	$e[(\lambda x.\, \exists y.\, y)/z]$ if z occurs in e only as $z[v]$

Fig. 18. Open lambdas (incomplete)

5 FOUNDATIONAL CALCULI

5.1 Untyped lambda calculus

$$
\begin{array}{rcl}
e & ::= & k \mid op \mid x \mid e_1\ e_2 \mid \lambda x.\, e \\
V & = & \mathbb{Z} + (V \to V) \\
\mathcal{E}[\![e]\!] & :: & Env \to V \\
\mathcal{E}[\![k]\!]\ \rho & = & k \\
\mathcal{E}[\![x]\!]\ \rho & = & \rho(x) \\
\mathcal{E}[\![e_1\ e_2]\!]\ \rho & = & (\mathcal{E}[\![e_1]\!]\ \rho)\ (\mathcal{E}[\![e_2]\!]\ \rho) \\
\mathcal{E}[\![\lambda x.\, e]\!]\ \rho & = & \lambda v.\, \mathcal{E}[\![e]\!]\ \rho[x := v]
\end{array}
$$

Problem: what is V? Standard answer: domain theory.

5.2 Simply typed lambda calculus in Martin-Löf Type Theory

$$
\begin{array}{rcl}
t & ::= & \mathit{Int} \mid t_1 \to t_2 \\
e & ::= & k \mid \mathit{op} \mid x \mid e_1\ e_2 \mid \lambda(x:t).\ e \\
\Gamma & ::= & \epsilon \mid \Gamma, x:t
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{T}[\![t]\!] & :: & \mathit{Type} \\
\mathcal{T}[\![\mathit{Int}]\!]\ \rho & = & \mathbb{Z} \\
\mathcal{T}[\![t_1 \to t_2]\!]\ \rho & = & \Pi(x : \mathcal{T}[\![t_1]\!]).\mathcal{T}[\![t_2]\!]
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{G}[\![\Gamma]\!] & :: & \mathit{dom}\ (\Gamma) \to \mathit{Type} \\
\mathcal{G}[\![\epsilon]\!] & = & \emptyset \\
\mathcal{G}[\![\Gamma], x:t & = & \mathcal{G}[\![\Gamma]\!], x :: \mathcal{T}[\![t]\!]
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{E}[\![\Gamma \vdash e : t]\!] & :: & \mathcal{G}[\![\Gamma]\!] \to \mathcal{T}[\![t]\!] \\
\mathcal{E}[\![\Gamma \vdash k : \mathit{Int}]\!]\ \rho & = & k \\
\mathcal{E}[\![\Gamma \vdash x : t]\!]\ \rho & = & \rho(x) \\
\mathcal{E}[\![\Gamma \vdash (e_1\ e_2) : t]\!]\ \rho & = & (\mathcal{E}[\![\Gamma \vdash e_1 : t_1 \to t]\!]\ \rho)(\mathcal{E}[\![\Gamma \vdash e_2 : t_1]\!]\ \rho) \\
 & & \text{where } \Gamma \vdash e_1 : t_1 \to t \text{ and } \Gamma \vdash e_2 : t_1 \\
\mathcal{E}[\![\Gamma \vdash (\lambda(x:t_1).\ e) : t_1 \to t_2]\!]\ \rho & = & \lambda\!\!\lambda(a \in \mathcal{T}[\![t_1]\!]).\ \mathcal{E}[\![\Gamma, x:t_1 \vdash e : t_2]\!]\ \rho[x \coloneqq a]
\end{array}
$$

Notes:

- This interpretation is only defined for well-typed terms.
- More standard to use nat instead of Int.
- $\Pi x : A.B$ is MLTT dependent function type.
- $x \in A \mapsto a$ is a MLTT function.
- The interpretation of a context Γ is type. This operation computes the type of ρ, a finite map from the variables in the domain of the context to values in $\mathcal{T}[\![\Gamma\ x]\!]$. In the abstraction case, we extend this finite map with a new binding for the variable x. In the variable case, we retrieve value of x from the map—this must be total because we know that $x \in \mathit{dom}\ \Gamma$.

5.3 Simply typed lambda calculus

$$
\begin{array}{rcl}
t & ::= & \mathit{Int} \mid t_1 \to t_2 \\
e & ::= & k \mid \mathit{op} \mid x \mid e_1\ e_2 \mid \lambda(x:t).\ e
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{PI} & :: & \mathit{Set}(A) \to \mathit{Set}(A \times B) \to \mathit{Set}(A \to B) \\
\mathit{PI}(A, R) & = & \{f \mid f \subseteq R,\ \mathit{isFunction}(f),\ \mathit{dom}(f) = A\}
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{T}[\![t]\!] & :: & \mathit{Set} \\
\mathcal{T}[\![\mathit{Int}]\!]\ \rho & = & \mathbb{Z} \\
\mathcal{T}[\![t_1 \to t_2]\!]\ \rho & = & \mathit{PI}(\mathcal{T}[\![t_1]\!],\ \mathcal{T}[\![t_1]\!] \times \mathcal{T}[\![t_2]\!])
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{E}[\![e]\!] & :: & \mathit{Env} \to \mathit{Set} \\
\mathcal{E}[\![k]\!]\ \rho & = & \{k\} \\
\mathcal{E}[\![x]\!]\ \rho & = & \{\rho(x)\} \\
\mathcal{E}[\![e_1\ e_2]\!]\ \rho & = & \{b \mid f \in \mathcal{E}[\![e_1]\!]\ \rho,\ a \in \mathcal{E}[\![e_2]\!]\ \rho,\ (a,b) \in f\} \\
\mathcal{E}[\![\lambda(x:t).\ e]\!]\ \rho & = & \mathit{PI}(A,\ \{(a,b) \mid a \in A,\ b \in \mathcal{E}[\![e]\!]\ (\rho[x \coloneqq a])\}) \\
 & & \text{where } A = \mathcal{T}[\![t]\!] \\
\mathcal{E}[\![\mathbf{fix}\ e]\!]\ \rho & = & \{a \mid f \in \mathcal{E}[\![e]\!]\ \rho,\ (a,a) \in f\}
\end{array}
$$

NOTES:

- $\mathcal{T}[\![t_1]\!] \times \mathcal{T}[\![t_2]\!]$ is a set-theoretic product of two sets.
- $isFunction(f)$ holds when f is a set of tuples, and for every $(x, y_1) \in f$ and $(x, y_2) \in f$, we have $y_1 = y_2$.

5.4 Timbda-0

A tiny calculus that captures some of the essence of Verse; especially, *terms and types are the same thing.*

$$
\begin{array}{rcl}
e & ::= & k \mid \mathbf{int} \mid x \mid e_1\ e_2 \\
& \mid & \lambda(x \coloneqq e_1).\ e_2 \\
& \mid & :e \\
& \mid & \mathbf{fail} \mid e_1 \mid e_2
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{E}[\![e]\!] & :: & Env \to Set \\
\mathcal{E}[\![k]\!]\ \rho & = & \{k\} \\
\mathcal{E}[\![x]\!]\ \rho & = & \{\rho(x)\} \\
\mathcal{E}[\![\mathbf{int}]\!]\ \rho & = & \{\{(a,a) \mid a \in \mathbb{Z}\}\} \\
\mathcal{E}[\![:e]\!]\ \rho & = & \{b \mid f \in \mathcal{E}[\![e]\!]\ \rho,\ (a,b) \in f\} \\
\mathcal{E}[\![e_1\ e_2]\!]\ \rho & = & \{b \mid f \in \mathcal{E}[\![e_1]\!]\ \rho,\ a \in \mathcal{E}[\![e_2]\!]\ \rho,\ (a,b) \in f\} \\
\mathcal{E}[\![\lambda(x \coloneqq e_1).\ e_2]\!]\ \rho & = & PI(A,\ \{(a,b) \mid a \in A,\ b \in \mathcal{E}[\![e_2]\!]\ (\rho[x \coloneqq a])\}) \\
 & & \text{where } A = \mathcal{E}[\![e_1]\!]\ \rho \\
\mathcal{E}[\![\mathbf{fail}]\!]\ \rho & = & \{\} \\
\mathcal{E}[\![e_1 \mid e_2]\!]\ \rho & = & \mathcal{E}[\![e_1]\!]\ \rho \cup \mathcal{E}[\![e_2]\!]\ \rho
\end{array}
$$

Fig. 19. The Timbda-0 calculus

We can translate STLC into Timbda 0.

$$
\begin{array}{rcl}
\mathcal{TT}[\![Int]\!] & = & :\mathbf{int} \\
\mathcal{TT}[\![t_1 \to t_2]\!] & = & \lambda(x \coloneqq \mathcal{TT}[\![t_1]\!]).\ \mathcal{TT}[\![t_2]\!] \\
 & & \\
\mathcal{TE}[\![k]\!] & = & k \\
\mathcal{TE}[\![x]\!] & = & x \\
\mathcal{TE}[\![e_1\ e_2]\!] & = & (\mathcal{TE}[\![e_1]\!])(\mathcal{TE}[\![e_2]\!]\ \rho) \\
\mathcal{TE}[\![\lambda(x:t).\ e]\!] & = & \lambda(x \coloneqq \mathcal{TT}[\![t]\!]).\ \mathcal{TE}[\![e]\!]
\end{array}
$$

Fig. 20. Translating STLC to Timbda-0

6 SEMANTICS

Domains

$$
\begin{array}{rlll}
v \in & \mathit{Val} & = \mathbf{I}(\mathbb{Z}) + \mathbf{F}(\mathbb{F}) + \mathbf{R}(\mathbb{R}) & \\
 & \mathbb{F} & = \mathcal{M}(\mathit{Val} \times \mathit{Val}) & \text{Invariants? e.g. disjoint domains} \\
 & \mathbb{R} & = \mathcal{M}(\mathit{Val} \times \mathit{Val}) & \text{Relations} \\
\rho \in & \mathit{Env} & = \mathit{Iden} \to \mathit{Val} & \text{Total: gives a value to every identifier} \\
\Delta \in & \mathit{ENV} & = \mathit{Set}(\mathit{Env}) & \\
\{x_1 \mapsto y_1, .., x_n \mapsto y_n\} \in & \mathit{Val} \rightharpoonup \mathit{Val} & = \mathit{Set}(\mathit{Val} \times \mathit{Val}) & \text{Partial functions}
\end{array}
$$

The constraint language

$$
\begin{array}{llll}
\text{Simple values} & a & ::= & v \mid x \mid \mathit{op}[a_1, .., a_n] \mid \langle a_0, \cdots, a_n \rangle \\
\text{Constraints} & \Delta & ::= & \{\!\{\, \mathit{relop}[a_1, .., a_n] \,\}\!\} \\
 & & \mid & \{\!\{\, x \mapsto y \in h \,\}\!\} \mid \{\!\{\, x \notin \mathbf{dom}(h) \,\}\!\} \quad \text{where } h \in \mathit{Val} \rightharpoonup \mathit{Val} \\
 & & \mid & \mathbf{empty} \mid \mathbf{univ} \\
 & & \mid & \Delta_1 \cup \Delta_2 \mid \Delta_1 \cap \Delta_2 \\
 & & \mid & \mathbf{compl}(\Delta) \quad \text{Complement} \\
 & & \mid & \Delta \setminus\setminus xs \quad \text{Hiding} \\
\text{Relational ops} & \mathit{relop} & ::= & = \mid < \mid \leqslant \mid \geqslant \mid > \\
\text{Value ops} & \mathit{op} & ::= & + \mid - \mid * \mid / \\
 & & \mid & \oplus \mid \boxplus \quad \text{Function and tuple concatenation resp.} \\
 & & \mid & \uplus \quad \text{Function/tuple union}
\end{array}
$$

Interpretation of simple values and constraints

$$
\begin{array}{rcll}
[\![\Delta]\!] & : & \mathit{ENV} & \\
[\![\mathbf{empty}]\!] & = & \{\} & \\
[\![\mathbf{univ}]\!] & = & \mathit{Env} & \text{Set of all environmen} \\
[\![\mathbf{compl}(\Delta)]\!] & = & \mathbf{compl}([\![\Delta]\!]) & \\
[\![\Delta \setminus\setminus xs]\!] & = & \{\rho \in \mathit{Env} \mid \exists \rho' \in [\![\Delta]\!]. \forall x \notin xs.\, \rho(x) = \rho'(x)\} & \\
[\![\Delta_1 \cup \Delta_2]\!] & = & [\![\Delta_1]\!] \cup [\![\Delta_2]\!] & \\
[\![\Delta_1 \cap \Delta_2]\!] & = & [\![\Delta_1]\!] \cap [\![\Delta_2]\!] & \\
[\![\{\!\{\, a_1 = a_2 \,\}\!\}]\!] & = & \{\rho \in \mathit{Env} \mid \mathcal{A}[\![a_1]\!]\rho = \mathcal{A}[\![a_2]\!]\rho\} & \\
[\![\{\!\{\, a_1 > a_2 \,\}\!\}]\!] & = & \{\rho \in \mathit{Env} \mid \mathcal{A}[\![a_1]\!]\rho > \mathcal{A}[\![a_2]\!]\rho\} & \text{...and similarly other } \mathit{relops} \\
[\![\{\!\{\, x \mapsto y \in h \,\}\!\}]\!] & = & \{\rho \in \mathit{Env} \mid \rho(x) \mapsto \rho(y) \in h\} & \\
[\![\{\!\{\, x \notin \mathbf{dom}(h) \,\}\!\}]\!] & = & \{\rho \in \mathit{Env} \mid \rho(x) \notin \mathbf{dom}(h)\} &
\end{array}
$$

$$
\begin{array}{rcll}
\mathcal{A}[\![a]\!] & : & \mathit{Env} \rightharpoonup \mathit{Val} & \\
\mathcal{A}[\![x]\!]\rho & = & \rho(x) & \\
\mathcal{A}[\![v]\!]\rho & = & v & \\
\mathcal{A}[\![a_1 + a_2]\!]\rho & = & \mathbf{I}(r_1 + r_2) & \text{if } \mathbf{I}(r_1) = \mathcal{A}[\![a_1]\!]\rho \text{ and } \mathbf{I}(r_2) = \mathcal{A}[\![a_2]\!]\rho \\
\mathcal{A}[\![a_1 \uplus a_2]\!]\rho & = & \mathbf{F}(r_1 \boxdot r_2) & \text{if } \mathbf{F}(r_1) = \mathcal{A}[\![a_1]\!]\rho,\ \mathbf{F}(r_2) = \mathcal{A}[\![a_2]\!]\rho,\ \mathbf{dom}(r_1) \cap \mathbf{dom}(r_2) = \emptyset \\
\mathcal{A}[\![a_1 \oplus a_2]\!]\rho & = & \mathbf{F}(r_1 \fatbslash r_2) & \text{if } \mathbf{F}(r_1) = \mathcal{A}[\![a_1]\!]\rho,\ \mathbf{F}(r_2) = \mathcal{A}[\![a_2]\!]\rho,\ \mathbf{dom}(r_1) \cap \mathbf{dom}(r_2) = \emptyset \\
\mathcal{A}[\![a_1 \boxplus a_2]\!]\rho & = & \mathbf{F}(\mathbf{tupcat}(r_1, r_2)) & \text{if } \mathbf{F}(r_1) = \mathcal{A}[\![a_1]\!]\rho,\ \mathbf{F}(r_2) = \mathcal{A}[\![a_2]\!]\rho,\ \mathbf{dom}(r_1) \cap \mathbf{dom}(r_2) = \emptyset \\
\mathcal{A}[\![\langle\rangle]\!]\rho & = & \mathbf{F}([\emptyset]) & \\
\mathcal{A}[\![\langle a_0, \cdots, a_n \rangle]\!]\rho & = & \mathbf{F}(\ [\{0 \mapsto \mathcal{A}[\![a_0]\!]\rho\}] \fatbslash \cdots \fatbslash [\{n \mapsto \mathcal{A}[\![a_n]\!]\rho\}]\ ) &
\end{array}
$$

Semantic functions

$$
\begin{array}{llcl}
\text{Binders of block} & \mathcal{I}[\![t]\!] & : & \mathit{Set}(\mathit{Iden}) \\
 & \mathcal{I}[\![t]\!] & = & \text{the outer } \exists\text{-bound variables in } t
\end{array}
$$

Fig. 21. Semantic domains and associated functions

The Verse computation type $\mathcal{M}(\cdot)$ comes equipped with the following operations

$\mathbf{empty}_m$	::	$\mathcal{M}(a)$	
$\mathbf{inj}$	::	$Set(a) \to \mathcal{M}(a)$	
$+\!\!+\!\!+_m$	::	$\mathcal{M}(a) \to \mathcal{M}(a) \to \mathcal{M}(a)$	Ordered choice
$\fatsemi$	::	$\mathcal{M}(a) \to \mathcal{M}(a) \to \mathcal{M}(a)$	Inner-set intersection
$\uplus$	::	$\mathcal{M}(a) \to \mathcal{M}(a) \to \mathcal{M}(a)$	Inner-set union
$\cup$	::	$\mathcal{M}(a) \to \mathcal{M}(a) \to \mathcal{M}(a)$	Unordered choice
$\mathbf{unionS}$	::	$Set(\mathcal{M}(a)) \to \mathcal{M}(a)$	Infinite unordered choice
$\mathbf{intersectS}$	::	$Set(\mathcal{M}(a)) \to \mathcal{M}(a)$	Infinite unordered intersection
$\mathbf{map}_m$	::	$(a \to b) \to \mathcal{M}(a) \to \mathcal{M}(b)$	
$\mathbf{map2}_m$	::	$(a \to b \to c) \to \mathcal{M}(a) \to \mathcal{M}(b) \to \mathcal{M}(c)$	
$\mathbf{fold}_m$	::	$(Set(a) \to b \to b) \to b \to \mathcal{M}(a) \to b$	Fold over choices
Π_m	:	$\mathcal{M}(a) \to \mathcal{M}(a,b) \to \mathcal{M}(\mathcal{M}(a,b))$	

Desirable laws

$$\mathbf{inj}(\{\}) = \mathbf{empty}_m$$
$$\mathbf{empty}_m \fatsemi s = \mathbf{empty}_m = s \fatsemi \mathbf{empty}_m$$
$$\mathbf{empty}_m +\!\!+\!\!+_m s = s = s +\!\!+\!\!+_m \mathbf{empty}_m$$
$$\mathbf{empty}_m \uplus s = s = s \uplus \mathbf{empty}_m$$
$$\mathbf{unionS}(\{\}) = \mathbf{empty}_m$$
$$\mathbf{intersectS}(\{\}) = \mathbf{empty}_m$$
$$\mathbf{fold}_m(k, z, \mathbf{empty}_m) = z$$
$$\mathbf{fold}_m(k, z, s +\!\!+\!\!+_m t) = \mathbf{fold}_m(k, \mathbf{fold}_m(k, z, t), s)$$
$$\mathbf{collapse}(\mathbf{inj}(s)) = s$$

Derived functions

$$\mathbf{mapS} :: (Set(a) \to Set(b)) \to \mathcal{M}(a) \to \mathcal{M}(b)$$
$$\mathbf{mapS}(f, s) = \mathbf{fold}_m(\ (\lambda x\, t.\, \mathbf{inj}(f(x)) +\!\!+\!\!+_m t),\ \mathbf{empty}_m,\ s\ )$$
$$(\ \backslash\backslash\backslash\ ) :: \mathcal{M}(Env) \to Set(Iden) \to \mathcal{M}(Env) \quad \text{Binds less tightly than } \fatsemi, \uplus \text{ etc}$$
$$s \backslash\backslash\backslash\ vs = \mathbf{mapS}(\ (\lambda \Delta.\, \Delta \backslash\backslash vs),\ s\ )$$
$$\mathbf{not} :: \mathcal{M}(Env) \to \mathcal{M}(Env)$$
$$\mathbf{not}(s) = \mathbf{inj}(\mathbf{compl}(\mathbf{collapse}(s)))$$
$$\mathbf{concat} :: [\mathcal{M}(a)] \to \mathcal{M}(a)$$
$$\mathbf{concat}(\ []\ ) = \mathbf{empty}_m$$
$$\mathbf{concat}(\ s : ss\ ) = s +\!\!+\!\!+_m \mathbf{concat}(ss)$$
$$\mathbf{collapse} :: \mathcal{M}(a) \to Set(a)$$
$$\mathbf{collapse}(s) = \mathbf{fold}_m(\cup, \{\}, s)$$
$$\mathbf{flatten} :: \mathcal{M}(Set(b)) \to \mathcal{M}(b)$$
$$\mathbf{flatten}(S) = \mathbf{mapS}(\bigcup,\ S)$$
$$\mathbf{flatMap} :: (a \to Set(b)) \to \mathcal{M}(a) \to \mathcal{M}(b)$$
$$\mathbf{flatMap}(f, S) = \mathbf{flatten}(\ \mathbf{map}_m(f, S)\ )$$

Fig. 22. The Verse computation type $\mathcal{M}(\cdot)$

$$
\begin{array}{rcl}
\textbf{prune} & :: & \mathcal{M}(a) \to \mathcal{M}(a) \\
\textbf{prune}(s) & = & \textbf{fold}_m(op, \textbf{empty}_m, s) \\
\multicolumn{3}{l}{\text{where}} \\
\multicolumn{3}{l}{\quad op :: \mathit{Set}(a) \to \mathcal{M}(a) \to \mathcal{M}(a)} \\
\multicolumn{3}{l}{\quad op(\Delta, \mathit{rest}) = \begin{cases} \mathit{rest} & \text{if } \Delta = \{\} \\ \textbf{inj}(\Delta) \mathbin{+\!\!+\!\!+_m} \mathit{rest} & \text{otherwise} \end{cases}}
\end{array}
$$

$$
\begin{array}{rcl}
\textbf{telescope} & :: & \mathit{Set}(\mathit{Iden}) \to \mathcal{M}(\mathit{Env}) \to \mathcal{M}(\mathit{Env}) \\
\textbf{telescope}(xs, S) & = & \textbf{unionS}\{\ \textbf{prune}(\ \textbf{inj}(\{\rho\} \setminus\setminus xs) \mathbin{\text{⍓}} S\ )\ \mid\ \rho \in \mathit{Env}\} \\
\textbf{telescope}(xs, S) & = & \text{Do we need a new operation on } \mathcal{M}(\cdot), e.g.\textbf{caseM}?
\end{array}
$$

$$
\begin{array}{rcl}
\textbf{index} & :: & \mathit{Iden} \to \mathcal{M}(\mathit{Env}) \to \mathcal{M}(\mathit{Env}) \quad \text{Number each choice with } i = 0,\ i = 1 \text{ etc} \\
\textbf{index}(i, s) & = & \textbf{fold}_m(op, \textbf{empty}_m, s)(0) \\
\multicolumn{3}{l}{\text{where}} \\
\multicolumn{3}{l}{\quad op :: \mathit{Set}(\mathit{Env}) \to \mathcal{M}(\mathit{Env}) \to \mathbb{Z} \to \mathcal{M}(\mathit{Env})} \\
\multicolumn{3}{l}{\quad op(\Delta, \mathit{rest})(k) = \textbf{inj}(\Delta \cap \{\!\!\{\ i = k\ \}\!\!\}) \mathbin{+\!\!+\!\!+_m} \mathit{rest}(k+1)}
\end{array}
$$

$$
\begin{array}{rcl}
\textbf{one} & :: & \mathit{Set}(\mathit{Iden}) \to \mathcal{M}(\mathit{Env}) \to \mathit{Set}(\mathit{Env}) \\
\textbf{one}(vs, S) & = & \textbf{fold}_m(\lambda\Delta \_ . \Delta,\ \{\},\ \textbf{telescope}(vs, S)) \\
\text{or}\quad \textbf{one}(vs, S) & = & \textbf{fold}_m(op, \{\}, S) \\
\multicolumn{3}{l}{\text{where}} \\
\multicolumn{3}{l}{\quad op :: \mathit{Set}(\mathit{Env}) \to \mathit{Set}(\mathit{Env}) \to \mathit{Set}(\mathit{Env})} \\
\multicolumn{3}{l}{\quad op(\Delta, \mathit{rest}) = \Delta \cup ((\textbf{compl}(\Delta \setminus\setminus vs)) \cap \mathit{rest})}
\end{array}
$$

Fig. 23. More derived functions for $\mathcal{M}(\cdot)$

Set semantics – a language with no "choice" operator

type $\mathcal{M}(a) = Set(a)$

Operations on sets

$$
\begin{array}{rcll}
\cup & :: & Set(a) \to Set(a) \to Set(a) & \text{Union} \\
\cap & :: & Set(a) \to Set(a) \to Set(a) & \text{Intersection} \\
\bigcup & :: & Set(Set(a)) \to Set(a) & \text{Big-union} \\
\bigcap & :: & Set(Set(a)) \to Set(a) & \text{Big-intersection} \\
\Pi & :: & Set(a) \to (a \to Set(b)) \to Set(a \rightharpoonup b) & \\
\Pi(s, F) & = & \{g \in a \rightharpoonup b \mid \mathbf{dom}(g) = s, \forall (x \mapsto y) \in g.\ y \in F(x)\} &
\end{array}
$$

Implementation of $\mathcal{M}(a) = Set(a)$

$$
\begin{array}{rcl}
\mathbf{empty}_m & = & \{\} \\
\mathbf{inj}(\Delta) & = & \Delta \\
s \mathbin{+\!\!+\!\!+_m} t & = & s \cup t \\
\mathbf{fold}_m(k, z, s) & = & k(s, z) \\
\overset{*}{\cap}(s, t) & = & s \cap t \\
\overset{*}{\cup}(s, t) & = & s \cup t \\
\cup(s, t) & = & s \cup t \\
\mathbf{unionS}(ss) & = & \bigcup ss \\
\mathbf{intersectS}(ss) & = & \bigcap ss \\
\Pi_m(s, F) & = & \Pi(s, F)
\end{array}
$$

Fig. 24. Set semantics, no choice

Sequence-of-set semantics
type $\mathcal{M}(a) = \mathcal{S}(Set(a))$

The Verse non-empty-sequence monad $\mathcal{S}(\cdot)$ comes equipped with the following operations

$$
\begin{array}{rcl}
(+\!\!+\!\!+_s) & :: & \mathcal{S}(a) \to \mathcal{S}(a) \to \mathcal{S}(a) \\
\mathbf{unit}_s & :: & a \to \mathcal{S}(a) \\
(\gg\!=_s) & :: & \mathcal{S}(a) \to (a \to \mathcal{S}(b)) \to \mathcal{S}(b) \\
\mathbf{fold}_s & :: & (a \to b \to b) \to b \to \mathcal{S}(a) \to b \\
\mathbf{dzip}_s & :: & (Set(a) \to a) \to Set(\mathcal{S}(a)) \to \mathcal{S}(a) \quad \text{Dodgy-zip of infinite set}
\end{array}
$$

Implementation of $\mathcal{M}(a) = \mathcal{S}(Set(a))$

$$
\begin{array}{rcl}
\mathbf{empty}_m & = & \mathbf{unit}_s(\{\}) \\
\mathbf{inj}(\Delta) & = & \mathbf{unit}_s(\Delta) \\
s +\!\!+\!\!+_m t & = & s +\!\!+\!\!+_s t \\
\uparrow (s,t) & = & s \gg\!=_s \lambda\Delta_1 . t \gg\!=_s \lambda\Delta_2 . \mathbf{unit}_s(\Delta_1 \cap \Delta_2) \\
\uplus (s,t) & = & s \gg\!=_s \lambda\Delta_1 . t \gg\!=_s \lambda\Delta_2 . \mathbf{unit}_s(\Delta_1 \cup \Delta_2) \\
\cup (s,t) & = & \mathbf{unionS}(\{s,t\}) \\
\mathbf{unionS}(ss) & = & \mathbf{dzip}_s(\bigcup, ss) \qquad \text{where } \bigcup :: Set(Set(a)) \to Set(a) \\
\mathbf{intersectS}(ss) & = & \mathbf{dzip}_s(\bigcap, ss) \qquad \text{where } \bigcap :: Set(Set(a)) \to Set(a) \\
\mathbf{map}_m(f,s) & = & s \gg\!=_s \lambda\Delta . \mathbf{unit}_s(\{f(x) \mid x \in \Delta\}) \\
\mathbf{map2}_m(f,s,t) & = & s \gg\!=_s \lambda\Delta_s . t \gg\!=_s \lambda\Delta_t . \mathbf{unit}_s(\{f(x,y) \mid x \in \Delta_s, y \in \Delta_t\}) \\
\mathbf{fold}_m(k,z,s) & = & \mathbf{fold}_s(k,z,s) \\
 & & \\
\Pi_m(\mathbf{unit}_s(\{\}), r) & = & \mathbf{unit}_s(\{\mathbf{empty}_m\}) \\
\Pi_m(\mathbf{unit}_s(\Delta_d), \mathbf{unit}_s(\Delta_r)) & = & \ldots todo \ldots \\
\Pi_m(d_1 +\!\!+\!\!+_s d_2, r) & = & \mathbf{map2}_m((+\!\!+\!\!+_m), \Pi_m(d_1,r), \Pi_m(d_2,r))
\end{array}
$$

Fig. 25. Sequence-of-set semantics

$$
\begin{array}{lrcll}
 & \mathcal{E}[\![t]\!]uv & : & \mathcal{M}(Env) & \\
\textsc{swild} & \mathcal{E}[\![\_]\!]uv & = & \mathbf{inj}\{\!\{\, u = v \,\}\!\} & \\
\textsc{svar} & \mathcal{E}[\![x]\!]uv & = & \mathbf{inj}\{\!\{\, u = v = x \,\}\!\} & \\
\textsc{sconst} & \mathcal{E}[\![k]\!]uv & = & \mathbf{inj}\{\!\{\, u = v = k \,\}\!\} & \\
\textsc{sprim} & \mathcal{E}[\![o]\!]uv & = & \mathbf{inj}\{\!\{\, u = v = o \,\}\!\} & \\
\textsc{sbind} & \mathcal{E}[\![x \coloneqq t]\!]uv & = & \mathbf{inj}\{\!\{\, x = v \,\}\!\} \fatsemi \mathcal{E}[\![t]\!]uv & \\
\textsc{stup} & \mathcal{E}[\![\mathbf{array}\{t_1, \cdots, t_n\}]\!]uv & = & (\mathbf{inj}\{\!\{\, u = \langle u_1, \cdots, u_n\rangle, v = \langle v_1, \cdots, v_n\rangle \,\}\!\} & \\
 & & & \quad \fatsemi \mathcal{E}[\![t_1]\!]u_1v_1 \fatsemi \ldots \fatsemi \mathcal{E}[\![t_n]\!]u_nv_n) & \\
 & & & \quad \backslash\!\backslash\!\backslash \{u_1, \ldots u_n, v_1 \ldots v_n\} & u_i, v_i \notin \mathsf{fvs}(t_1 \ldots t_n, u, v) \\
\textsc{sfail} & \mathcal{E}[\![\mathbf{fail}]\!]uv & = & \mathbf{empty}_m & \\
\textsc{schoice} & \mathcal{E}[\![t_1 \mid t_2]\!]uv & = & \mathcal{B}[\![t_1]\!]uv +\!\!+\!\!+_m \mathcal{B}[\![t_2]\!]uv & \\
\textsc{sunify} & \mathcal{E}[\![t_1 = t_2]\!]uv & = & \mathcal{E}[\![t_1]\!]uv \fatsemi \mathcal{E}[\![t_2]\!]uv & \\
\textsc{ssquig} & \mathcal{E}[\![t_1 \rightsquigarrow t_2]\!]uv & = & (\mathcal{E}[\![t_1]\!]uw \fatsemi \mathcal{E}[\![t_2]\!]wv) \backslash\!\backslash\!\backslash \{w\} & w \notin \mathsf{fvs}(t_1, t_2, u, v) \\
\textsc{sseq} & \mathcal{E}[\![t_1;\ t_2]\!]uv & = & \mathcal{C}[\![t_1]\!] \fatsemi \mathcal{E}[\![t_2]\!]uv & \\
\textsc{swhere} & \mathcal{E}[\![t_1 \ \mathbf{where}\ t_2]\!]uv & = & \mathcal{E}[\![t_1]\!]uv \fatsemi \mathcal{C}[\![t_2]\!] & \\
\textsc{sdotdot} & \mathcal{E}[\![t_1 \mathinner{..} t_2]\!]uv & = & \mathbf{inj}\{\!\{\, u = v \,\}\!\} \fatsemi \mathcal{E}[\![t_1]\!]p_1q_1 \fatsemi \mathcal{E}[\![t_2]\!]p_2q_2 \fatsemi & \\
 & & & \quad \mathbf{unionS}\{\ n \in \mathbb{N} \parallel \mathbf{concat}[\mathbf{inj}\{\!\{\, v = q_1 + i, i \leqslant q_2 - q_1 \,\}\!\} \mid i \leftarrow [0..n]]\ \} & \\
 & & & \quad \backslash\!\backslash\!\backslash \{p_1, q_1, p_2, q_2\} & p_1, q_1, p_2, q_2 \notin \mathsf{fvs}(t_1, t_2, u, v) \\
\textsc{sapp} & \mathcal{E}[\![t_1[t_2]]\!]uv & = & \mathbf{inj}\{\!\{\, u = v \,\}\!\} \fatsemi \mathcal{E}[\![t_1]\!]fg \fatsemi \mathcal{E}[\![t_2]\!]pq \fatsemi \mathcal{F}[\![g[q]]\!]v & \\
 & & & \quad \backslash\!\backslash\!\backslash \{f, g, p, q\} & f, g, p, q \notin \mathsf{fvs}(t_1, t_2, u, v) \\
\textsc{stype} & \mathcal{E}[\![:t]\!]uv & = & \mathcal{E}[\![t]\!]pq \fatsemi \mathcal{F}[\![q[u]]\!]v \backslash\!\backslash\!\backslash \{p, q\} & p, q \notin \mathsf{fvs}(t, u, v)
\end{array}
$$

$$
\begin{array}{rcll}
\mathcal{B}[\![t]\!]uv & : & \mathcal{M}(Env) & \\
\mathcal{B}[\![t]\!]uv & = & \mathcal{E}[\![t]\!]uv \backslash\!\backslash\!\backslash \mathcal{I}[\![t]\!] & \\
\mathcal{C}[\![t]\!] & : & \mathcal{M}(Env) & \\
\mathcal{C}[\![t]\!] & = & \mathcal{E}[\![t]\!]pq \backslash\!\backslash\!\backslash \{p, q\} & p, q \notin \mathsf{fvs}(t) \\
\mathcal{F}[\![f[x]]\!]r & : & \mathcal{M}(Env) & \\
\mathcal{F}[\![f[x]]\!]r & = & \mathbf{unionS}\{\ \mathbf{mapS}(\ \lambda prs.\{\!\{\, f = v,\ x \mapsto r \in prs \,\}\!\}, ff\ ) & \\
 & & \quad \mid v \in Val,\ ff \in \mathcal{M}(Val, Val) & \\
 & & \quad ,\ v = \mathbf{F}(ff) \vee v = \mathbf{R}(ff)\ \} &
\end{array}
$$

Fig. 26. Denotational semantics of Essential Verse, Part 1

$$\textsc{sif}\ \mathcal{E}[\![\mathbf{if}(t_0)\{t_1\}\,\mathbf{else}\,\{t_2\}]\!]uv \;=\; (\mathbf{inj}(\Delta) \fatsemi \mathcal{B}[\![t_1]\!]uv \setminus\!\setminus\!\setminus\ xs) \cup (\mathbf{inj}(\mathbf{compl}(\Delta \setminus\!\setminus\!\setminus\ xs)) \fatsemi \mathcal{B}[\![t_2]\!]uv)$$
$$\mathbf{where}\ xs = \mathcal{I}[\![t_0]\!],\ \Delta = \mathbf{one}(xs, \mathcal{C}[\![t_0]\!])$$

$$\textsc{sindex}\ \mathcal{E}[\![\mathbf{choiceIndex}\{t\}]\!]uv$$
$$= \mathbf{index}(\ u,\ \mathbf{telescope}(\mathcal{I}[\![t]\!],\ \mathcal{E}[\![t]\!]jv \setminus\!\setminus\!\setminus\ \{j\})\ ) \qquad j \notin \mathsf{fvs}(t,u,v)$$

$$\textsc{srel}\ \mathcal{E}[\![\mathbf{rel}(at)\{bt\}]\!]hf$$
$$= \mathbf{inj}\{\ \rho \in Env \qquad \{u_a, v_a, u_b, v_b\} \notin \mathsf{fvs}(at, bt, h, f)$$
$$\mid \mathbf{let}\ S :: \mathcal{M}(Env) = (\mathbf{inj}(\{\rho\}) \setminus\!\setminus\!\setminus\ \{u_a, v_a, \mathcal{I}[\![at]\!]\} \fatsemi \mathcal{E}[\![at]\!]u_a v_a) \setminus\!\setminus\!\setminus\ \{u_b, v_b, \mathcal{I}[\![bt]\!]\} \fatsemi \mathcal{E}[\![bt]\!]u_b v_b)$$
$$,\ \rho(h) = \rho(f) = \mathbf{R}(\mathbf{map}_m(\ \lambda\rho.(\rho(u_a), \rho(v_b)),\ \mathbf{prune}(S)\ ))\ \}$$

$$\textsc{sfun}\ \mathcal{E}[\![\mathbf{fun}(at)\langle q\rangle\langle\omega\rangle\{bt\}]\!]hf$$
$$= \mathbf{unionS}\{\ \mathbf{flatMap}(keep(\rho), funs(\rho))\ \mid\ \rho \in Env\ \}$$
$$\text{where}$$
$$S_a :: \mathcal{M}(Env) = \mathcal{E}[\![at]\!]xw$$
$$S_b :: \mathcal{M}(Env) = \mathcal{E}[\![bt]\!]jz$$
$$funs : Env \to \mathcal{M}(\mathcal{M}(Val, Env)) \qquad \#\ (x,(w,j,z))$$
$$funs(\rho) = \quad \Pi_m(\ domvs, rngvs\ )$$
$$\text{where}$$
$$dom : \mathcal{M}(Env)$$
$$dom = (\mathbf{inj}\{\rho\} \setminus\!\setminus\!\setminus\ \mathcal{I}[\![at]\!] \setminus\!\setminus\!\setminus\ \{x, w\}) \fatsemi S_a$$
$$domvs : \mathcal{M}(Val)$$
$$domvs = \mathbf{map}_m(\ \lambda\rho.\ \rho(x),\ dom\ )$$
$$rngvs :: \mathcal{M}(Val, Env)$$
$$rngvs = \mathbf{unionS}\{\ \mathbf{intersectS}\{\ (\mathbf{inj}(\{\rho\}) \setminus\!\setminus\!\setminus\ \mathcal{I}[\![bt]\!]) \fatsemi S_b \mid \rho \in \mathbf{collapse}(dom), \rho(x) = xv\ \} \mid xv \in \mathbf{collapse}(domvs)\ \}$$
$$keep :: Env \to \mathcal{M}(Val, Env) \to Set(Env) \qquad \#\ \text{Singleton or empty}$$
$$keep(\rho)(fun) = \begin{cases} \{\rho\} & \text{if } pick(\rho, fun) \\ \{\} & \text{otherwise} \end{cases}$$
$$pick :: Env \to \mathcal{M}(Val, Env) \to Bool$$
$$pick(\rho, fun) = \mathbf{case}\ q\ \mathbf{of}$$
$$\langle\mathbf{closed}\rangle \to fr = fm\ \wedge hr = hm$$
$$\langle\mathbf{open}\rangle \to\ \forall v \in \mathbf{dom}(fm).\ fm\ (v) = fr\ (v)\ \wedge$$
$$\forall v \in \mathbf{dom}(hm).\ hm\ (v) = hr\ (v)\ \wedge$$
$$\forall v \notin \mathbf{dom}(fm).\ hr\ (v) = fr\ (v)$$
$$\mathbf{where}$$
$$pf = \mathbf{prune}(fun)$$
$$fm, hm :: Val$$
$$fm = \mathbf{F}(\mathbf{map}_m(\lambda(x,\rho').\ (x, \rho'(z)),\ pf))$$
$$hm = \mathbf{F}(\mathbf{map}_m(\lambda(\_,\rho').\ (\rho'(w), \rho'(j)),\ pf))$$
$$fr, hr :: Val$$
$$fr = \rho(f)$$
$$hr = \rho(h)$$

Fig. 27. Denotational semantics of Essential Verse, Part 2

$$
\begin{array}{lrcll}
 & \mathcal{E}[\![t]\!] & : & \mathcal{M}(Env) & \\
\textsc{swild} & \mathcal{E}[\![\_]\!] & = & \mathbf{inj}\{\!|\, \mathbf{u}=\mathbf{v} \,|\!\} & \\
\textsc{svar} & \mathcal{E}[\![x]\!] & = & \mathbf{inj}\{\!|\, \mathbf{u}=\mathbf{v}=x \,|\!\} & \\
\textsc{sconst} & \mathcal{E}[\![k]\!] & = & \mathbf{inj}\{\!|\, \mathbf{u}=\mathbf{v}=k \,|\!\} & \\
\textsc{sprim} & \mathcal{E}[\![o]\!] & = & \mathbf{inj}\{\!|\, \mathbf{u}=\mathbf{v}=o \,|\!\} & \\
\textsc{sbind} & \mathcal{E}[\![x \coloneqq t]\!] & = & \mathbf{inj}\{\!|\, x=\mathbf{u} \,|\!\} \fatsemi \mathcal{E}[\![t]\!] & \\
\textsc{stup} & \mathcal{E}[\![\mathbf{array}\{t_1,\cdots,t_n\}]\!] & = & (\mathbf{inj}\{\!|\, u=\langle u_1,\cdots,u_n\rangle, v=\langle v_1,\cdots,v_n\rangle \,|\!\} & \\
 & & & \fatsemi \mathcal{D}[\![t_1]\!]u_1v_1 \fatsemi \ldots \fatsemi \mathcal{D}[\![t_n]\!]u_nv_n) & \\
 & & & \backslash\!\backslash\!\backslash \{u_1,..u_n,v_1..v_n\} & u_i, v_i \notin \mathsf{fvs}(t_1..t_n) \\
\textsc{sfail} & \mathcal{E}[\![\mathbf{fail}]\!] & = & \mathbf{empty}_m & \\
\textsc{schoice} & \mathcal{E}[\![t_1 \mid t_2]\!] & = & \mathcal{B}[\![t_1]\!] +\!\!+_m \mathcal{B}[\![t_2]\!] & \\
\textsc{sunify} & \mathcal{E}[\![t_1 = t_2]\!] & = & \mathcal{E}[\![t_1]\!] \fatsemi \mathcal{E}[\![t_2]\!] & \\
\textsc{ssquig} & \mathcal{E}[\![t_1 \rightsquigarrow t_2]\!] & = & (\mathcal{D}[\![t_1]\!]\mathbf{u}w \fatsemi \mathcal{D}[\![t_2]\!]w\mathbf{v}) \backslash\!\backslash\!\backslash \{w\} & w \notin \mathsf{fvs}(t_1,t_2,u,v) \\
\textsc{sseq} & \mathcal{E}[\![t_1;\ t_2]\!] & = & \mathcal{C}[\![t_1]\!] \fatsemi \mathcal{E}[\![t_2]\!] & \\
\textsc{swhere} & \mathcal{E}[\![t_1\ \mathbf{where}\ t_2]\!] & = & \mathcal{E}[\![t_1]\!] \fatsemi \mathcal{C}[\![t_2]\!] & \\
\textsc{sdotdot} & \mathcal{E}[\![t_1 \mathinner{..} t_2]\!] & = & \mathbf{inj}\{\!|\, \mathbf{u}=\mathbf{v} \,|\!\} \fatsemi \mathcal{D}[\![t_1]\!]p_1q_1 \fatsemi \mathcal{D}[\![t_2]\!]p_2q_2 \fatsemi & \\
 & & & \mathbf{unionS}\{\, n \in \mathbb{N} \parallel \mathbf{concat}[\mathbf{inj}\{\!|\, \mathbf{v}=q_1+i, i \leqslant q_2-q_1 \,|\!\} \mid i \leftarrow [0..n]] \,\} & \\
 & & & \backslash\!\backslash\!\backslash \{p_1,q_1,p_2,q_2\} & p_1,q_1,p_2,q_2 \notin \mathsf{fvs}(t_1,t_2) \\
\textsc{sapp} & \mathcal{E}[\![t_1[t_2]]\!] & = & \mathbf{inj}\{\!|\, \mathbf{u}=\mathbf{v} \,|\!\} \fatsemi \mathcal{D}[\![t_1]\!]fg \fatsemi \mathcal{D}[\![t_2]\!]pq \fatsemi \mathcal{F}[\![g[q]]\!]\mathbf{v} & \\
 & & & \backslash\!\backslash\!\backslash \{f,g,p,q\} & f,g,p,q \notin \mathsf{fvs}(t_1,t_2) \\
\textsc{stype} & \mathcal{E}[\![:t]\!] & = & (\mathcal{D}[\![t]\!]pq \fatsemi \mathcal{F}[\![q[u]]\!]\mathbf{v}) \backslash\!\backslash\!\backslash \{p,q\} & p,q \notin \mathsf{fvs}(t)
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{B}[\![t]\!] & : & \mathcal{M}(Env) \\
\mathcal{B}[\![t]\!] & = & \mathcal{E}[\![t]\!] \backslash\!\backslash\!\backslash \mathcal{I}[\![t]\!] \\
\mathcal{C}[\![t]\!] & : & \mathcal{M}(Env) \\
\mathcal{C}[\![t]\!] & = & \mathcal{E}[\![t]\!] \backslash\!\backslash\!\backslash \{\mathbf{u},\mathbf{v}\} \\
\mathcal{D}[\![t]\!]pq & : & \mathcal{M}(Env) \\
\mathcal{D}[\![t]\!]pq & = & (\mathcal{E}[\![t]\!] \fatsemi \{\!|\, \mathbf{u}=p, \mathbf{v}=q \,|\!\}) \backslash\!\backslash\!\backslash \{\mathbf{u},\mathbf{v}\} \\
\mathcal{F}[\![f[x]]\!]r & : & \mathcal{M}(Env) \\
\mathcal{F}[\![f[x]]\!]r & = & \mathbf{unionS}\{\, \mathbf{mapS}(\ \lambda prs.\{\!|\, f=v,\ x \mapsto r \in prs \,|\!\}, ff\ ) \\
 & & \quad \mid v \in Val,\ ff \in \mathcal{M}(Val, Val) \\
 & & \quad ,\ v = \mathbf{F}(ff) \vee v = \mathbf{R}(ff) \,\}
\end{array}
$$

Fig. 28. Semantics without explicit u,v

$$\textsc{sif}\ \mathcal{E}[\![\mathbf{if}(t_0)\{t_1\}\mathbf{else}\{t_2\}]\!]uv \;=\; (\mathbf{inj}(\Delta) \upharpoonright \mathcal{B}[\![t_1]\!]uv \setminus\!\setminus\!\setminus xs) \cup (\mathbf{inj}(\mathbf{compl}(\Delta \setminus\!\setminus\!\setminus xs)) \upharpoonright \mathcal{B}[\![t_2]\!]uv)$$
$$\mathbf{where}\ xs = \mathcal{I}[\![t_0]\!],\ \Delta = \mathbf{one}(xs, \mathcal{C}[\![t_0]\!])$$

$$\textsc{sindex}\ \mathcal{E}[\![\mathbf{choiceIndex}\{t\}]\!]uv$$
$$= \mathbf{index}(\ u,\ \mathbf{telescope}(\mathcal{I}[\![t]\!],\ \mathcal{E}[\![t]\!]jv \setminus\!\setminus\!\setminus \{j\})\ ) \qquad j \notin \mathsf{fvs}(t,u,v)$$

$$\textsc{srel}\ \mathcal{E}[\![\mathbf{rel}(at)\{bt\}]\!]hf$$
$$= \mathbf{inj}\{\ \rho \in Env \qquad \{u_a, v_a, u_b, v_b\} \notin \mathsf{fvs}(at, bt, h, f)$$
$$\quad \mid \mathbf{let}\ S :: \mathcal{M}(Env) = (\mathbf{inj}(\{\rho\}) \setminus\!\setminus\!\setminus \{u_a, v_a, \mathcal{I}[\![at]\!]\} \upharpoonright \mathcal{E}[\![at]\!]u_a v_a) \setminus\!\setminus\!\setminus \{u_b, v_b, \mathcal{I}[\![bt]\!]\} \upharpoonright \mathcal{E}[\![bt]\!]u_b v_b)$$
$$\quad ,\ \rho(h) = \rho(f) = \mathbf{R}(\mathbf{map}_m(\ \lambda\rho.(\rho(u_a), \rho(v_b)),\ \mathbf{prune}(S))\ )\ \}$$

$$\textsc{sfun}\ \mathcal{E}[\![\mathbf{fun}(at)\langle q\rangle\langle\omega\rangle\{bt\}]\!]hf$$
$$= \mathbf{unionS}\{\ \mathbf{flatMap}(keep(\rho), funs(\rho))\ \mid\ \rho \in Env\ \}$$
where
$$S_a :: \mathcal{M}(Env) = \mathcal{E}[\![at]\!]xw$$
$$S_b :: \mathcal{M}(Env) = \mathcal{E}[\![bt]\!]jz$$
$$funs : Env \to \mathcal{M}(\mathcal{M}(Val, Env)) \qquad \#\ (x,(w,j,z))$$
$$funs(\rho) = \quad \Pi_m(\ domvs, rngvs\ )$$
where
$$dom : \mathcal{M}(Env)$$
$$dom = (\mathbf{inj}\{\rho\} \setminus\!\setminus\!\setminus \mathcal{I}[\![at]\!] \setminus\!\setminus\!\setminus \{x, w\}) \upharpoonright S_a$$
$$domvs : \mathcal{M}(Val)$$
$$domvs = \mathbf{map}_m(\ \lambda\rho.\ \rho(x),\ dom\ )$$
$$rngvs :: \mathcal{M}(Val, Env)$$
$$rngvs = \mathbf{unionS}\{\ \mathbf{intersectS}\{\ (\mathbf{inj}(\{\rho\}) \setminus\!\setminus\!\setminus \mathcal{I}[\![bt]\!]) \upharpoonright S_b \mid \rho \in \mathbf{collapse}(dom), \rho(x) = xv\ \} \mid xv \in \mathbf{collapse}(domvs)\ \}$$
$$keep :: Env \to \mathcal{M}(Val, Env) \to Set(Env) \qquad \#\ \text{Singleton or empty}$$
$$keep(\rho)(fun) = \begin{cases} \{\rho\} & \text{if } \rho(f) = \mathbf{F}(\mathbf{map}_m(\lambda(x,\rho').\ (x, \rho'(z)),\ \mathbf{prune}(fun))) \\ & \quad\ \rho(h) = \mathbf{F}(\mathbf{map}_m(\lambda(\_,\rho').\ (\rho'(w), \rho'(j)),\ \mathbf{prune}(fun))) \\ \{\} & \text{otherwise} \end{cases}$$

Fig. 29. Denotational semantics of Essential Verse, Part 2