Chapter 5 (draft): Data and Pattern Matching in the Cast Language

Mark Lemay

February 10, 2022

inline document

path var,
x_p
assertion index,
k
assertion assumption,

kin	$::=$	$k = left \mid k = right$	

assertion maps,

$kmap$	$::=$	$\overline{kin,}$	

casts under assumption,

$kcast$	$::=$	$\overline{kmap, p;}$	

path exp.,

p, p'	$::=$	x_p	path variable
	$\mid$	$Cong_{k \Rightarrow a}^{\overline{q}:\mathit{\Gamma} \approx \mathit{\Gamma}', p:A \approx B}$	congruence
	$\mid$	$Assert_{o,\ell,a:A,b:B}^{\overline{q}:\mathit{\Gamma} \approx \mathit{\Gamma}', p:A \approx B}$	concrete assumption
	$\mid$	$\overline{p;}$	concatenated paths
	$\mid$	p^{-1}	reverse a path
	$\mid$	$inTC_i\, p$	extract the $i$type argument from a data type constructor
	$\mid$	$inC_i\, p$	extract the $i$type argument from a data constructor

cast pattern,

$patc$	$::=$	$x \mid \left(d\, \overline{patc}\right) :: x_p$	

cast expressions,

a, b, A, B	$::=$	x	
	$\mid$	L	
	$\mid$	$a :: L$	cast
	$\mid$	$\star$	
	$\mid$	$(x : A) \to B$	
	$\mid$	$\mathsf{fun}\, f\, x \Rightarrow b$	
	$\mid$	$b\, a$	
	$\mid$	D_Δ	type cons.
	$\mid$	d_Δ	data cons.
	$\mid$	$\mathsf{case}\, \overline{a,}\, \left\{\overline{\mid \overline{patc \Rightarrow} b} \overline{\mid \overline{patc'_\ell}}\right\}$	data elim.
	$\mid$	$!_{kcast}$	force blame
	$\mid$	$a :: kcast$	cast
	$\mid$	$\{a \sim_{k,p} b\}$	assert same

observations,

o	$::=$	...	
	$\mid$	$o.App[a]$	application
	$\mid$	$o.TCon_i$	type cons. arg.
	$\mid$	$o.DCon_i$	data cons. arg.
	$\mid$	$inEx_{\overline{patc}}[\overline{a}]$	in-exhaustive pattern match

Lumps

L, l	$::=$	$\{\star\}$	
	$\mid$	$\left\{\overline{a}, \overline{b \sim c}, \overline{x_p}_{L'}\right\}$	
	$\mid$	$Arg\ L$	
	$\mid$	$Bod\,[a]\ \ L$	
		...	

contexts,

$\mathit{\Gamma}$	$::=$	.	
	$\mid$	$x : A$	
	$\mid$	$x_p : A \approx B$	

1 Judgment Forms

$$\Gamma \vdash a : A$$

$$\Gamma \vdash a \sqsupseteq a' : A'$$

$$\Gamma \vdash l \ \mathbf{ok}$$

$$\Gamma \vdash l \ \mathbf{connected}$$

$$\Gamma \vdash a \equiv a' : A$$

2 Definitional equality

Assume a congruent equivelence that

- respects evaluation. $\Gamma \vdash a \rightsquigarrow a' : A$ implies $\Gamma \vdash a \equiv a' : A$
- associates trivial casts with uncast terms. $\Gamma \vdash a : A$ implies $\Gamma \vdash a \equiv a :: \left\{A_{\{\star\}}\right\} \ : A$
- associates casts at the same endpoints. $\Gamma \vdash a \equiv a' : A$, $\Gamma \vdash a :: L \ : B$, $\Gamma \vdash a :: L' \ : B$ implies $\Gamma \vdash a :: L \ \equiv \ a' :: L' \ : \ A$

3 Typing Rules

$$\frac{x : A \ \in \Gamma}{\Gamma \vdash x : A}$$

$$\frac{x_p : a \approx b \ \in \Gamma}{\Gamma \vdash x_p : a \approx b} ??typing??$$

$$\frac{}{\Gamma \vdash \star : \star}$$

$$\frac{\begin{array}{c}\Gamma \vdash a : A \\ \Gamma \vdash L \ \mathbf{ok} \\ \Gamma \vdash L \sqsupseteq A : \star \\ \Gamma \vdash L \sqsupseteq B : \star\end{array}}{\Gamma \vdash a :: L \ : B}$$

...

$$\frac{\Gamma \vdash A : \star \quad \Gamma, x : A \vdash B : \star}{\Gamma \vdash \prod x : A.B \ : \star}$$

...

4 Lumping Rules

$$\frac{}{\Gamma \vdash \{\star\} \ \mathbf{ok}}$$

$$\frac{\begin{array}{c}L \ \mathbf{connected} \\ L = \{..._{L'}\} \\ \Gamma \vdash L' \ \mathbf{ok} \\ \forall a. \Gamma \vdash L \sqsupseteq a : A, \quad L' \sqsupseteq A\end{array}}{\Gamma \vdash L_{L'} \ \mathbf{ok}}$$

every endpont it connected and every type is connected. Revise,

- $\forall A$?
- instead of endpoints, consider elements
- add typing conditions on elements

Example 1. $\left\{1 \sim 2, S\left\{0 \sim 5_{\{\mathbb{N}\}}\right\}, 6 \sim 7_{\{\mathbb{N}\}}\right\}$ **ok**
since $S\left\{0 \sim 5_{\{\mathbb{N}\}}\right\} \sqsupseteq 1$,$S\left\{0 \sim 5_{\{\mathbb{N}\}}\right\} \sqsupseteq 6$
every endpoint is connected up to **normalization** and **casts**
evey type of every endpoint is connected

Will need to be a little flexble with connections

Example 2. $\left\{True \sim 1 :: \{\mathbb{N} \sim \mathbb{B}\}, 1 :: \{\mathbb{N} \sim \mathbb{B}\} \sim False_{\{\mathbb{B}\}}\right\}$ **ok**

5 Endpoint Rules

5.1 incorrect lump endpoints

we want to select the endpoints of a lump

$$\frac{\begin{array}{c}\Gamma \vdash a \in L \\ \Gamma \vdash a : A\end{array}}{\Gamma \vdash L \sqsupseteq a : A}$$

but each term my contain sub terms

$$\frac{\begin{array}{c}\Gamma \vdash a \in L \\ \Gamma \vdash a \sqsupseteq a' : A\end{array}}{\Gamma \vdash L \sqsupseteq a' : A}$$

and we need that each one is fuged with it's own cast

$$\frac{\begin{array}{c}\Gamma \vdash a \in L \\ L = \{..._{L'}\} \\ \Gamma \vdash a \sqsupseteq a' : A \\ \Gamma \vdash L' \sqsupseteq B : \star\end{array}}{\Gamma \vdash L \sqsupseteq a' :: L' \ : \ B}$$

5.2 lump endpoints

$$\frac{\begin{array}{c}\Gamma \vdash a \in L \\ L = \{..._{L'}\} \\ \Gamma \vdash a \sqsupseteq a' : A \\ \Gamma \vdash L' \sqsupseteq B : \star\end{array}}{\Gamma \vdash L \sqsupseteq a' :: L' \ : \ B}$$

$$\frac{\begin{array}{c}\Gamma \vdash a \sim b \in L \\ \Gamma \vdash a \sqsupseteq a' : A \\ ...\end{array}}{\Gamma \vdash L \sqsupseteq a' : A}$$

$$\frac{\begin{array}{c}\Gamma \vdash b \sim a \in L \\ \Gamma \vdash a \sqsupseteq a' : A \\ ...\end{array}}{\Gamma \vdash L \sqsupseteq a' : A}$$

$$\frac{\begin{array}{c}\Gamma \vdash x_p \in L \\ \Gamma \vdash x_p : a \approx b \\ \Gamma \vdash a \sqsupseteq a' : A \\ ...\end{array}}{\Gamma \vdash L \sqsupseteq a' : A} ????$$

$$\frac{\begin{array}{c}\Gamma\vdash x_p\in L\\ \Gamma\vdash x_p : b\approx a\\ \Gamma\vdash a\sqsupseteq a' : A\\ ....\end{array}}{\Gamma\vdash L\sqsupseteq a' : A}\,????$$

5.3 other endpoints

we would like the endpoint relation to otherwise be reflexive

$$\frac{\Gamma\vdash a : A}{\Gamma\vdash a\sqsupseteq a : A}$$

however this means $\{1\sim 2\}\sqsupseteq\{1\sim 2\} : \mathbb{N}$. which can lead to types with unclear meanings, like $Vec\ \{1\sim 2\}$.
other rules mimick thier typing judgments

$$\frac{\Gamma\vdash a\sqsupseteq a' : A\quad \Gamma\vdash a'\equiv a'' : A}{\Gamma\vdash a\sqsupseteq a'' : A}$$

$$\frac{\Gamma\vdash A\sqsupseteq A' : \star\quad \Gamma, x : A'\vdash B\sqsupseteq B' : \star}{\Gamma\vdash \prod x : A.B\ \sqsupseteq\ \prod x : A'.B'\ :\ \star}$$

$$\frac{\Gamma, f : (x : A)\to B, x : A\vdash b\sqsupseteq b' : B'}{\Gamma\vdash \mathsf{fun}\, f\, x\Rightarrow b'\ :\ (x : A)\to B'}$$

$$\frac{\Gamma\vdash b\ \sqsupseteq\ b' : (x : A')\to B'\quad \Gamma\vdash a\sqsupseteq\ a' : A'}{\Gamma\vdash b\, a\ \sqsupseteq\ b'\, a'\ :\ B'\,[x\coloneqq a']}$$

...

Example 3. $\left\{7\sim True_{\{\mathbb{N}\sim\mathbb{B}\}}\right\}\sqsupseteq\ 7 : \mathbb{N}$
$\left\{7\sim True_{\{\mathbb{N}\sim\mathbb{B}\}}\right\}\sqsupseteq\ True : \mathbb{B}$
$\left\{7\sim True_{\{\mathbb{N}\sim\mathbb{B}\}}\right\}\sqsupseteq\ True :: \{\mathbb{N}\sim\mathbb{B}\} : \mathbb{N}$
$\left\{7\sim True_{\{\mathbb{N}\sim\mathbb{B}\}}\right\}\sqsupseteq\ 7 :: \{\mathbb{N}\sim\mathbb{B}\} : \mathbb{B}$
$\left\{7\sim True_{\{\mathbb{N}\sim\mathbb{B}\}}\right\}\sqsupseteq\ 7 :: \{\mathbb{N}\sim Tuesday, Tuesday\sim\mathbb{B}\} : \mathbb{B}$

...

Example 4. $\left\{rep\sim not_{\{(x:\mathbb{N})\to Vec\,x\sim\mathbb{B}\to\mathbb{B}\}}\right\}\left\{7\sim True_{\{\mathbb{N}\sim\mathbb{B}\}}\right\}\sqsupseteq\ rep\, 7 : Vec\, 7$
$\left\{rep\sim not_{\{(x:\mathbb{N})\to Vec\,x\sim\mathbb{B}\to\mathbb{B}\}}\right\}\left\{7\sim True_{\{\mathbb{N}\sim\mathbb{B}\}}\right\}\sqsupseteq\ not\, True : \mathbb{B}$
$\left\{rep\sim not_{\{(x:\mathbb{N})\to Vec\,x\sim\mathbb{B}\to\mathbb{B}\}}\right\}\left\{7\sim True_{\{\mathbb{N}\sim\mathbb{B}\}}\right\}\not\sqsupseteq not\, 7 : \mathbb{B}$

The typeing restriction protects type boundries.
revise
$\left\{rep\sim not_{\{(x:\mathbb{N})\to Vec\,x\sim\mathbb{B}\to\mathbb{B}\}}\right\}\left\{7\sim True_{\{\mathbb{N}\sim\mathbb{B}\}}\right\}?\sqsupseteq?\ not\ (7 :: \{\mathbb{N}\sim\mathbb{B}\}) : \mathbb{B}$
yes

6 CBV

evaluation will preserve the types of well typed terms, the endpoints, and the OKness of lumps

$$\frac{}{\star\sim\star\ \rightsquigarrow\ \star}$$

Revise this so the syntax is more "formal".
Casts can be bumped out of the way....

$$\frac{}{\{..., a :: L, ..._{L'}\}\ \rightsquigarrow\ \{..., a, ..._{L'\cup L}\}}$$

$$\frac{}{\{..., a :: L\sim b, ..._{L'}\}\ \rightsquigarrow\ \{..., a\sim b, ..._{L'\cup L}\}}$$

$$\frac{}{\{..., a \sim b :: L, ..._{L'}\} \ \rightsquigarrow \ \{..., a \sim b, ..._{L' \cup L}\}}$$

need to formalize the endpoints of and Arged Lump

can constrain the type

$$\frac{}{Arg \left\{\prod x : A.B \sim_{\ell,o} \prod x : A'.B', ..., \prod x : A''.B'', ..., x_p, ..._L\right\} \ \rightsquigarrow \ \left\{A \sim_{\ell,o.Arg} A', ..., A'', ..., Arg\ \{x_{pL}\}, ..._{\{\star\}}\right\}}$$

Injecting the lump variable into a {} seems a little hacky.

$$\frac{}{Bod\,[a] \left\{\prod x : A.B \sim_{\ell,o} \prod x : A'.B', ..., \prod x : A''.B'', ..., x_p\right\} \ \rightsquigarrow \ \left\{B\,[a] \sim_{\ell,o.Bod[a]} B'\,[a]\,, ..., B''\,[a]\,, ..., Bod\,[a]\ \{x_{pL}\}\,, ..._{\{\star\}}\right\}}$$

...

7 Blame

...

8 Properties

Lump substituiton accross types must be handled carefully

9 Progress

...

10 Preservation

Part I

Old stuff - Ignore

path var,			
x_p			
assertion index,			
k			
assertion assumption,			
kin	$::=$	$k = left \mid k = right$	
assertion maps,			
$kmap$	$::=$	$\overline{kin,}$	
casts under assumption,			
$kcast$	$::=$	$\overline{kmap, p;}$	
path exp.,			
p, p'	$::=$	x_p	path variable
	$\mid$	$Cong_{k \Rightarrow a}^{\overline{q}:\Gamma\approx\Gamma', p:A\approx B}$	congruence
	$\mid$	$Assert_{o,\ell,a:A,b:B}^{\overline{q}:\Gamma\approx\Gamma', p:A\approx B}$	concrete assumption
	$\mid$	$\overline{p;}$	concatenated paths
	$\mid$	p^{-1}	reverse a path
	$\mid$	$inTC_i\, p$	extract the $i$type argument from a data type constructor
	$\mid$	$inC_i\, p$	extract the $i$type argument from a data constructor
cast pattern,			
$patc$	$::=$	$x \mid \left(d\,\overline{patc}\right) :: x_p$	
cast expression,			
$a...$	$::=$	$...$	
		$\prod x : p_B^A.C$	
	$\mid$	D_Δ	type cons.
	$\mid$	d_Δ	data cons.
	$\mid$	$\mathsf{case}\,\overline{a,}\,\left\{\overline{\mid \overline{patc \Rightarrow} b \mid \overline{patc'_\ell}}\right\}$	data elim.
	$\mid$	$!_{kcast}$	force blame
	$\mid$	$a :: kcast$	cast
	$\mid$	$\{a \sim_{k,p} b\}$	assert same
observations,			
o	$::=$	$...$	
	$\mid$	$o.App[a]$	application
	$\mid$	$o.TCon_i$	type cons. arg.
	$\mid$	$o.DCon_i$	data cons. arg.
	$\mid$	$inEx_{\overline{patc}}[\overline{a}]$	in-exhaustive pattern match

11 Judgment forms

Γ	$\vdash$	p	$:$	$a : A$	$\approx_{\overline{q}:\Gamma'}$	$b : B$	pathing
Γ	$\vdash$	p	$:$	$a : A$	$\approx_{\Gamma'}$	$b : B$	pathing?
Γ^2	$\vdash_k$	a	$:$	$a : A$	$\approx$	$b : B$	pathing

operations

12 subst

!!!!

13 extended typing rules

structural rules

$$\frac{\Gamma \vdash p : a : A \approx_{-:\Gamma'} b : B}{\overline{q} \begin{array}{c}\Gamma \\ \Gamma'\end{array} \vdash_k \{a \sim_{k,p} b\} : a : A \approx b : B} ??$$

$$\frac{x : p_B^A \in \Gamma}{\Gamma \vdash_k x \ : x : A \approx x : B} ??$$

$$\frac{\Gamma, x : p_B^A \vdash_k C : \qquad C^L : \star \approx C^R : \star}{\Gamma \vdash_k \prod x : p_B^A . C \ : \prod x : A.C^L : \star \approx \prod x : B.C^R : \star} ??$$

$$\frac{\begin{array}{c}\Gamma \vdash_k b \ : \ b^L : \prod x : A^L.C^L \approx b^R : \prod x : A^R.C^R \\ \Gamma \vdash_k a \ : \ a^L : A^L \approx a^R : A^R\end{array}}{\Gamma \vdash_k b \ a \ : \ b^L \ a^L : C^L \left[a^L\right] \approx b^R \ a^R : C^R \left[a^R\right]} ??$$

$$\frac{\Gamma, x : refl_A^A \vdash_k b \ : \ b^L : B^L \approx b^R : B^R}{\Gamma \vdash_k \lambda x.b \ : \ b^L \ \prod x : A.C^L \approx b^R : \prod x : A.C^R} ...??$$

$$\frac{\begin{array}{c}\Gamma \vdash_k a \ : \ a^L : A^L \approx a^R : A^R \\ \Gamma \Uparrow\vdash p : A^L : - \approx_{-:\Gamma\uparrow} B^L : ... \\ \Gamma \Downarrow\vdash p : A^R : - \approx_{-:\Gamma\downarrow} B^R : ...\end{array}}{\Gamma \vdash_k a :: \begin{array}{c} k = L, p \\ k = R, q\end{array} \ : \ a^L :: p : B^L \approx a^R :: q : B^R} ...??$$

assert...
data...
conv...

14 pathing rules

$$\frac{\overline{q} \begin{array}{c}\Gamma \\ \Gamma'\end{array} \vdash_k a \ : \ a^L : A^L \approx a^R : A^R}{\Gamma \vdash Cong_{k \Rightarrow a}^{\overline{q}:\Gamma \approx \Gamma', ?:A \approx B} : a^L : A^L \approx_{\overline{q}:\Gamma'} a^R : A^R} ??$$

$$\frac{\begin{array}{c}\Gamma \vdash p \ : \ a : A \approx_{?:\Gamma'} b : B \\ \Gamma' \vdash q \ : \ b : B' \approx_{?:\Gamma''} c : C \\ \Gamma' \vdash r \ : \ B : \star \approx_{?:\Gamma'} B' : \star\end{array}}{\Gamma \vdash p;_r q \ : \ a : A \approx_{?:\Gamma'} c : C} ??$$

$$\frac{x_p : a : A \approx_{?:\Gamma'} b : B \in \Gamma}{\Gamma \vdash x_p : a : A \approx_{?:\Gamma'} b : B} ??$$

$$\frac{\Gamma \vdash p : a : A \approx_{?:\Gamma'} b : B}{\Gamma' \vdash p^{-1} : b : B \approx_{?^{-1}:\Gamma} a : A} ??$$

assert...
data...
conv...

15 cbv

$$\frac{\begin{array}{c} \mathit{\Gamma} \vdash b \ : \ b^L : B^L \approx b^R : B^R \\ \mathit{\Gamma} \vdash a \ : \ a^L : A^L \approx a^R : A^R \\ \mathit{\Gamma} \Uparrow\vdash p : b^L : B^L \approx_{-:\mathit{\Gamma}\uparrow} b^{L\prime} : \prod x : A^L.C^L \\ \mathit{\Gamma} \Downarrow\vdash q : b^R : B^R \approx_{-:\mathit{\Gamma}\downarrow} b^{R\prime} : \prod x : A^R.C^R \\ \mathit{\Gamma} \Uparrow\vdash p \ \mathit{Slice} \ p', \ p^{A^L}, \ p^{C^L} \\ \mathit{\Gamma} \Uparrow\vdash p' : b^L : B^L \approx_{-:\mathit{\Gamma}\uparrow} d^L : \prod x : A^{L\prime}.C^{L\prime} \\ \mathit{\Gamma} \Uparrow\vdash p^{A^L} : A^L : \star \approx_{-:\mathit{\Gamma}\uparrow} A^{L\prime} : \star \\ \mathit{\Gamma} \uparrow, x : A^L \vdash p^{C^L} : C^{L\prime} : \star \approx_{-:\mathit{\Gamma},x:A^{L\prime},} C^L : \star \\ \mathit{\Gamma} \Downarrow\vdash q \ \mathit{Slice} \ q', \ q^{A^R}, \ q^{C^R} \\ ... \end{array}}{\begin{array}{c} \mathit{\Gamma} \vdash_k \left(b :: \begin{array}{c} k = L, p \\ k = R, q \end{array} \right) a \\ \rightsquigarrow \left(b :: \begin{array}{c} k = L, p' \\ k = R, q' \end{array} \right) \left(a :: \begin{array}{c} k = L, p^{A^L} \\ k = R, p^{A^R} \end{array} \right) :: \begin{array}{c} k = L, p^{C^L} \left[a :: p^{A^L} \right] \\ k = R, q^{C^R} \left[a? \right] \end{array} \\ : ? : C^L \left[a^L \right] \approx ? : C^R \left[a^R \right] \end{array}} \ ...??$$

but will be weird if a sub path wiggles into another ctx! which seems to be protected by the ctx annotation. requires this awkward identity, $C^L \left[a :: p^{A^L} ::? \right] \equiv C^L \left[a^L \right]$ where ? is magically extracted by the path

Part II
Old Formalisms

16 Cast Language Pattern Matching

Several of the constructs familiar form the last chapter are extended. Patterns are extended with path variable. In addition to the expected branches the `case` construct now explicitly contains a collection of unmatched patterns that will allow for a static warning, and if reached, a runtime error.

In this thesis we have taken an extremely extensional perspective, terms are only different if an observation recognizes a difference. For instance, functions $\lambda x \Rightarrow x + 1 = \lambda x \Rightarrow 1 + x$ should be equatable without proof, even though they are usually definitionally distinct. Therefore we will only blame inequality across functions if two functions that were asserted to be equal return different observations for "the same" input. Tracking that two functions should be equal becomes complicated, the system must be sensible under context, functions can take other higher order inputs, and function terms can be copied freely.

Example back to pattern matching?

The cleanest way I could find to encode a dynamic check for function equality, was with a new term level construct, $\{a \sim_{k,o,\ell} b\}$ [1]. This assertion that two terms are the same is written as $\{a \sim_{k,o,\ell} b\}$ and will evaluate $a$and b in parallel until both a and b evaluate to a head constructor. If the head constructor is different the term will get stuck with the information for blame. If the constructor is the same it will commute out of the term. For instance, $\{\lambda x \Rightarrow x + 1 \sim_{k,o,\ell} \lambda x \Rightarrow 1 + x\} \rightsquigarrow_* \lambda x \Rightarrow \left\{x + 1 \sim_{k,o.App[x],\ell} 1 + x\right\}$.

Since this same construct seems the best way to handle functions, we will use it for all runtime equality assertions. For instance, $\{(\lambda x \Rightarrow S\,x)\,Z \sim_{k,o,\ell} 2 + 2\} \rightsquigarrow_* \{S\,Z \sim_{k,o,\ell} S\ (1 + 2)\} \rightsquigarrow_* S\ \{Z \sim_{k,o.DCon_0,\ell} 1 + 2\} \rightsquigarrow_* S\ \{Z \sim_{k,o.DCon_0,\ell} S\,2\}$. We compute past the first S constructor and blame the predecessor for not being equal.

Move later?

To control the use of this construct, assumptions will be bound into Assert paths. For instance, $\mathit{Assert}_{\ k \Rightarrow 1 + \{2 \sim_{k,o,\ell} 3\}}$ will represent a user defined assertion that $3 \approx 4$. Binding the $\{\sim\}$ to a specific assert forces the $\{\sim\}$ to only appear in paths, they cannot appear unbound in the empty context. Additionally the binding allows the assumptions to be precisely tracked when paths contain terms that contain paths that contain terms. For technical reasons, Asserts also hold evidence that the types of the 2 endpoints match, For instance in $\mathit{Assert}_{p\ k \Rightarrow \{true \sim_{k,o,\ell} 3\}}\ , p : \mathbb{B} \approx \mathbb{N}$. While separating user assumptions into casts, Asserts, and $\{\sim\}$, is more complicated then in Chapter 3, we have a

[1] it would also be possible to extend the system with meta variables, though this seems harder to formalize

path var,
x_p
assertion index,
k
assertion assumption,
$kin \quad ::= \quad k = left \mid k = right$
assertion maps,
$kmap \quad ::= \quad \overline{kin,}$
casts under assumption,
$kcast \quad ::= \quad \overline{kmap, p;}$
path exp.,

p, p'	$::=$	x_p	path variable
	$\mid$	$Assert_{p\ k \Rightarrow C}$	concrete assumption
	$\mid$	$\overline{p;}$	concatenated paths
	$\mid$	$cong_{x \Rightarrow a} p$	congruence
	$\mid$	p^{-1}	reverse a path
	$\mid$	$inTC_i\, p$	extract the $i$type argument from a data type constructor
	$\mid$	$inC_i\, p$	extract the $i$type argument from a data constructor
	$\mid$	$ty\, p$	derive a path between types

cast pattern,
$patc \quad ::= \quad x \mid \left(d\,\overline{patc}\right) :: x_p$
cast expression,

$a...$	$::=$	$...$	
	$\mid$	D_Δ	type cons.
	$\mid$	d_Δ	data cons.
	$\mid$	$\mathsf{case}\,\overline{a,}\ \left\{\overline{\mid \overline{patc} \Rightarrow b}\overline{\mid \overline{patc'_\ell}}\right\}$	data elim.
	$\mid$	$!_{kcast}$	force blame
	$\mid$	$a :: kcast$	cast
	$\mid$	$\{a \sim_{k,o,\ell} b\}$	assert same

observations,

o	$::=$	$...$	
	$\mid$	$o.App[a]$	application
	$\mid$	$o.TCon_i$	type cons. arg.
	$\mid$	$o.DCon_i$	data cons. arg.
	$\mid$	$inEx_{\overline{patc}}[\overline{a}]$	in-exhaustive pattern match

extend H ctxs

Figure 1: Cast Language Data

(the empty path)	written	$refl$		
$a :: kmap, p; kmap', q; ...$	written	$a :: \begin{array}{ll} kmap, & p \\ kmap', & q \\ ... & ... \end{array}$		
$a :: \begin{array}{ll} kmap, k = left, & p \\ kmap, k = right, & p \\ kmap', k = left, & q \\ kmap, k = right, & q \\ ... & ... \end{array}$	written	$a :: \begin{array}{ll} kmap, & p \\ kmap', & q \\ ... & ... \end{array}$	when	k is irrelevant
D_Δ	written	D	when	the telescope is clear from context
d_Δ	written	d	when	the telescope is clear from context
$Assert_{refl\ k \Rightarrow C}$	written	$Assert_{\ k \Rightarrow C}$		

Figure 2: Surface Language Abbreviations

clearer interpretation of assertions that only hold other assertions, here the binding assumption is simply not used, $Assert_{p\ -\Rightarrow C}$.

Unfortunately this dynamic assertion complicates other aspects of the system. Specifically,

- How do same assertions interact with casts? For instance, $\{1 :: Bool \sim_{k,o,\ell} 2 :: Bool\}$.
- How do sameness assertions cast check? This is difficult, because there is no requirement that a user asserted equality is of the same type. For instance what type should the term $\{1 \sim_{k,o,\ell} True\}$ have?

Since there will only ever be a finite number of assumptions, we can give each assumption a unique index k and consider all computations and judgments point-wise for every different combinations of ks. We will extend the notion of cast so different casts are possible for every assignment of ks in scope. So

$\{1 :: Bool \sim_{k,o,\ell} 2 :: Bool\} \rightsquigarrow_* \{1 \sim_{k,o,\ell} 2 :: Bool\} :: k = left, Bool \rightsquigarrow_* \{1 \sim_{k,o,\ell} 2\} :: \begin{matrix} k = right, Bool \\ k = left, Bool \end{matrix} = \{1 \sim_{k,o,\ell} 2\} :: Bool$

where we allow syntactic sugar to summarize the cast when they are the same over all branches (Figure 2).

We will also index typing judgments by the choice of k in scope so that, $k = left \vdash \{1 \sim_{k,o,\ell} True\} : Nat$ and $k = right \vdash \{1 \sim_{k,o,\ell} True\} : Bool$.

Now we must consider how patterns would evaluate under assumptions. The original inspiration was to allow abstraction over casts as represented by a path, but now casts contain a bundle of paths each indexed by assumptions, that may or may not have the same start and endpoint. Luckily, we can maintain the operational behavior by allowing uniform substitution into path variables.

For simplicity of the formalization, we will require that $kmap$s always uniquely map every k index in scope. We will also assume that $kcast$ handles all possible mappings of ks in scope.

Substitution is outlined in table 3. The function $\lfloor - \rfloor_{k=-}$ filters a term along k taking it out of scope. For instance $\lfloor\{7 \sim_{k,o,\ell} True\}\rfloor_{k=left} = 7$ and $\left\lfloor 3 :: \begin{matrix} k = left & j = left & Bool \\ k = left & j = right & Nat \\ k = right & j = left & String \\ k = right & j = right & Unit \end{matrix} \right\rfloor_{k=right} = 3 :: \begin{matrix} j = left & String \\ j = right & Unit \end{matrix}$. The function $\lceil - \rceil^k$ puts an assumption k into scope extending it in every cast. For instance, $\left\lceil 3 :: \begin{matrix} j = left & String \\ j = right & Unit \end{matrix} \right\rceil^k = 3 :: \begin{matrix} j = left & k = left & String \\ j = left & k = right & String \\ j = right & k = left & Unit \\ j = right & k = right & Unit \end{matrix}$ and $\lceil !_{x_p} \rceil^k = !_{\begin{matrix} k = left & x_p \\ k = right & x_p \end{matrix}}$. The subscript $kcast_{kmap}$ selects the appropriate assumption from the $kcast$. For instance $\begin{matrix} k = left & j = left & x_p \\ k = left & j = right & refl \\ k = right & j = left & y_p \\ k = right & j = right & x_p \end{matrix}_{k=right, j=left} = y_p$. Since we every choice in scope is handled, this result always exists.

17 Cast Value, Blame, and Reductions

We need to extend the notion of value, Blame and call-by-value reduction from Chapter 3.

The **Blame** relation from Chapter 3 are simplified via the sameness assertion4.

There are two new sources of blame from the `case` construct. The cast language records every "unhandled" branch and if a scrutinee hits one of those branches the case will be blamed for in-exhaustiveness [2]. If a scrutinee list primitively contradicts the pattern coverage via the !**Match** judgment blame will be extracted from the scrutinee. Since our type system will ensure complete coverage (based only on constructors) if a scrutinee escapes the complete pattern match in an empty context, it must be that there was a blamable cast to the head constructor.

As expected, the new syntax to explicitly trigger blame, !, will trigger blame when reached.

Say more about how blame behaves over terms and paths?

We have elided most of the structural rules that extract blame from terms, paths, and casts. We have left the structural rule for explicit blame for emphasis.

[2]This runtime error is conventional in ML style languages, and is even how Agda handles incomplete matches.

$$
\begin{array}{ccc}
\mathsf{case}\,\overline{b},\ \left\{\overline{|\,\overline{patc}\Rightarrow c}|\,\overline{patc'\Rightarrow !_{\ell}}\right\} & [x:=a]= & \mathsf{case}\,\overline{b\,[x:=a]}\,,\ \left\{\overline{|\,\overline{patc}\Rightarrow c\,[x:=a]}|\,\overline{patc'\Rightarrow !_{\ell}}\right\}\\
!_{kcast} & [x:=a]= & !_{kcast[x:=a]}\\
b::kcast & [x:=a]= & b\,[x:=a]::kcast\,[x:=a]\\
\{b\sim_{k,o,\ell}c\} & [x:=a]= & \left\{b\left[x:=\lfloor a\rfloor_{k=left}\right]\sim_{k,o[x:=a],\ell}c\left[x:=\lfloor a\rfloor_{k=right}\right]\right\}\\
\ldots & \ldots & \ldots\\
x_p & [x:=a]= & x_p\\
Assert_{p\ k\Rightarrow C} & [x:=a]= & Assert_{p[x:=a]\ k\Rightarrow C\left[x:=\lceil a\rceil^{k}\right]}\\
\overline{p\circ} & [x:=a]= & \overline{p\,[x:=a]\circ}\\
cong_{y\Rightarrow b}p & [x:=a]= & cong_{y\Rightarrow b[x:=a]}p\,[x:=a]\\
p^{-1} & [x:=a]= & p\,[x:=a]^{-1}\\
inTC_i\,p & [x:=a]= & inTC_i\ (p\,[x:=a])\\
inC_i\,p & [x:=a]= & inC_i\ (p\,[x:=a])\\
ty\,p & [x:=a]= & ty\ (p\,[x:=a])\\
\overline{kmap,p;} & [x:=a]= & \overline{kmap,p\left[x:=\lfloor a\rfloor_{kmap}\right];}\\
\ldots & & \ldots\\
x_p & [x_p:=p]= & p\\
\mathsf{case}\,\overline{a},\ \left\{\overline{|\,\overline{patc}\Rightarrow c}|\,\overline{patc'\Rightarrow !_{\ell}}\right\} & [x_p:=kcast]= & \mathsf{case}\,\overline{a\,[x_p:=kcast]}\,,\ \left\{\overline{|\,\overline{patc}\Rightarrow c\,[x_p:=kcast]}|\,\overline{patc'_{\ell}}\right\}\\
!_{kcast} & [x_p:=kcast]= & !_{kcast[x_p:=kcast]}\\
a::kcast' & [x_p:=kcast]= & a\,[x_p:=kcast]::kcast'\,[x_p:=kcast]\\
\{b\sim_{k,o,\ell}c\} & [x_p:=kcast]= & \left\{b\left[x:=\lfloor kcast\rfloor_{k=left}\right]\sim_{k,o[x:=a],\ell}c\left[x:=\lfloor kcast\rfloor_{k=right}\right]\right\}\\
\ldots & \ldots & \ldots\\
\overline{kmap,p;} & [x_p:=kcast]= & \overline{kmap,p\,[x_p:=kcast_{kmap}]\,;}
\end{array}
$$

Review this

Figure 3: Cast Language Data Sub

$$
\frac{a\ \textbf{whnf}\quad b\ \textbf{whnf}\quad \textbf{head}\,a\neq\textbf{head}\,b}{\textbf{Blame}\ \ell\, o\ \{a\sim_{k,o,\ell}b\}}
$$

$$
\frac{\overline{a}\ \textbf{Match}\ \overline{patc'}_j}{\textbf{Blame}\ \ell\ inEx_{\overline{patc'}_j}[\overline{a}]\ \mathsf{case}\,\overline{a},\ \left\{\overline{|\,\overline{patc}\Rightarrow b}|\,\overline{patc'\Rightarrow !_{\ell_j}}\right\}}
$$

fix syntax?

$$
\frac{\overline{a}\ !\textbf{Match}\ \ell\, o\,\overline{|\,\overline{patc'}_j}}{\textbf{Blame}\ \ell\, o\ \mathsf{case}\,\overline{a},\ \left\{\overline{|\,\overline{patc}\Rightarrow -}\right\}}
$$

$$
\frac{!_{kcast}\quad \textbf{Blame}\ \ell\, o\ kcast}{\textbf{Blame}\ \ell\, o\ !_{kcast}}
$$

Figure 4: Selection of Cast Language Blame

REDO THIS

$$\frac{\textbf{Blame}\ \ell\ o\ p}{\left(d'\ \overline{patc}\right) :: p\ !\textbf{Match}\ \ell\ o\ \left(d\ \overline{patc}\right) :: p}$$

This figure is not very helpful?

Figure 5: Cast Language Blame

just put them in

The value forms of the language must also be extended, values are presented in 6. As before type level values must have all type level casts reduced, term level values are allowed casts as long as they are plausible and in **whnf** .

This extension to the syntax induces many more reduction rules. We include a summery of selected reduction rules in 7. We do not show the value restrictions to avoid clutter[3]. The important properties of reduction are

- Paths reduce into a stack of zero or more $Assert_{k \Rightarrow A}$s
- Sameness assertions emit observably consistent constructors, and record the needed observations
- Sameness assertions will get stuck on inconsistent constructors
- Casts can commute out of sameness assertions with proper index tracking
- function application can commute around *kcasts* , similar to Chapter 3, but will keep k assumptions properly indexed

Matching is defined in 8. Note that uncast terms are equivalent to refl cast terms.

double check paths are fully applied when needed

The Cast language extension defined in this chapter is fairly complex. Though all the meta-theory of this section is plausible, we have not fully formalized it, and there is a potential that some subtle errors exist. To be as clear as possible about the uncertainty around the meta-theory proposed in this chapter, I will list what would normally be considered theorems and lemmas as conjectures.

weird plac
note. add
or back m

Conjecture 5. *There is a suitable definitional equality* $\equiv$ *, overloaded to all syntactic constructs, such that*
$\equiv$ is an equivalence
$a \rightsquigarrow_* b$ and $a' \rightsquigarrow_* b$ implies $a \equiv a'$
$p \rightsquigarrow_* q$ and $p' \rightsquigarrow_* q$ implies $p \equiv p'$
$kcast \rightsquigarrow_* kcast''$ and $kcast' \rightsquigarrow_* kcast''$ implies $kcast \equiv kcast''$
if **Head** $a \neq$ **Head** b *then* $a \not\equiv b$
if $kmap$ and $kmap'$ have consistent assignments then $kmap \equiv kmap'$

settle on a
for head

supports s

18 Cast System and Pathing

The cast system needs to maintain the consistency of well cast terms and also well typed paths. But unlike in Chapter 3 each judgment indexed by choices of k.

The typing and pathing judgments are listed in 9. Pathing judgments record the endpoint of paths with the $\approx$ symbol.

technically speaking, telescopes should generalize to the different syntactic classes

We now conjecture the core lemmas that could be used to prove cast soundness

Conjecture 6. *substitution of cast terms preserves cast*
equivalently the following rule is admissible

$$\frac{H\,K \vdash a : A \quad x : A \in H \quad H\,K \vdash b : B}{H\,[x := a]\,K \vdash b\,[x := a] : B\,[x := a]}$$

[3]there are also multiple ways to lay them out. For instance we could evaluate paths left to right or right to left.

$$\frac{}{\star\ \mathbf{Val}}$$

$$\frac{}{(x:A)\to B\ \mathbf{Val}}$$

$$\frac{kcast\ \mathbf{Val} \quad \forall Assert_{k\Rightarrow(x:A)\to B} \in kcast}{(fun\, f\, x \Rightarrow a) :: kcast\ \mathbf{Val}}$$

clean up notation above

$$\frac{\overline{a}\ \mathbf{Val}}{D\,\overline{a}\ \mathbf{Val}}$$

$$\frac{\overline{a}\ \mathbf{Val} \quad kcast\ \mathbf{Val} \quad \exists D.\forall Assert_{k\Rightarrow D\bar{b}} \in kcast}{(d\,\overline{a}) :: kcast\ \mathbf{Val}}$$

clean up notation above

$$\frac{p\ \mathbf{Val} \quad C\ \mathbf{Val} \quad C \neq \star}{Assert_{p\ k\Rightarrow C}\ \mathbf{Val}}$$

$$\frac{p\ \mathbf{Val} \quad q\ \mathbf{Val}}{p; q\ \mathbf{Val}}$$

$$\frac{}{refl\ \mathbf{Val}}$$

$$\frac{p\ \mathbf{Val}}{kmap, p\ \mathbf{Val}}$$

$$\frac{\forall kmap, p \in kcast,\ \ kmap, p\,\mathbf{Val}}{kcast\ \mathbf{Val}}$$

Figure 6: Selection of Cast Language Values

path reductions

$$\frac{}{Assert_{-\Rightarrow\star} \rightsquigarrow refl}$$

$$\frac{}{cong_{x\Rightarrow a}\left(Assert_{p\ \ k\Rightarrow c}\right) \rightsquigarrow Assert_{k\Rightarrow a[x:=castl(c,ty\ \ p)]}}$$

Assert is refl since cong is const over paths

given suitable condition on the kcast

$$\frac{}{cong_{x\Rightarrow a} refl \rightsquigarrow refl}$$

$$\frac{}{cong_{x\Rightarrow a}\left(p;q\right) \rightsquigarrow \left(cong_{x\Rightarrow a[x:=cast(x,leftty\ \ q)]} p\right);\left(cong_{x\Rightarrow a} q\right)}$$

this is a mess since now every path needs its endpoints!

$$\frac{}{\left(Assert_{k\Rightarrow C}\right)^{-1} \rightsquigarrow Assert_{k\Rightarrow \mathbf{Swap}_k C}}$$

$$\frac{}{refl^{-1} \rightsquigarrow refl}$$

$$\frac{}{\left(q\circ p\right)^{-1} \rightsquigarrow p^{-1}\circ q^{-1}}$$

$$\frac{}{inTC_i\left(p\circ Assert_{k\Rightarrow D\overline{a}}\right) \rightsquigarrow inTC_i\left(p\right)\circ Assert_{k\Rightarrow a_i}}$$

$$\frac{}{inTC_i\left(refl\right) \rightsquigarrow refl}$$

$$\frac{}{inC_i\left(p\circ Assert_{k\Rightarrow (d\overline{a})::kcast}\right) \rightsquigarrow inC_i\left(p\right)\circ Assert_{k\Rightarrow a_i}}$$

given suitable condition on the kcast

$$\frac{}{inC_i\left(refl\right) \rightsquigarrow refl}$$

structural rules

$$\frac{C \rightsquigarrow C'}{Assert_{k\Rightarrow C} \rightsquigarrow Assert_{k\Rightarrow C'}}$$

$$\frac{q \rightsquigarrow q'}{p\circ q \rightsquigarrow p\circ q'}$$

$$\frac{p \rightsquigarrow p'}{p\circ q \rightsquigarrow p'\circ q}$$

assumption reductions

$$\frac{p \rightsquigarrow p'}{\begin{array}{c} kcast \\ \overline{kin}, p; \\ kcast' \end{array} \rightsquigarrow \begin{array}{c} kcast \\ \overline{kin}, p'; \\ kcast' \end{array}}$$

$$\frac{a \rightsquigarrow a'}{a :: kcast \rightsquigarrow a' :: kcast}$$

$$\frac{kcast \rightsquigarrow kcast'}{a :: kcast \rightsquigarrow a :: kcast'}$$

term reductions

$$\frac{p \rightsquigarrow p'}{!_p \rightsquigarrow !_{p'}}$$

swap, pattern on the left?

record substitutions

$$\frac{}{a\ \mathbf{Match}\ x}$$

$$\frac{\overline{a}\ \mathbf{Match}\ \overline{patc}}{d\,\overline{a}\ \mathbf{Match}\ \left(d\,\overline{patc}\right) :: x_p}$$

$$\frac{\overline{a}\ \mathbf{Match}\ \overline{patc}}{\left(d\,\overline{a}\right) :: kcast\ \mathbf{Match}\ \left(d\,\overline{patc}\right) :: x_p}$$

$$\frac{\overline{a}\ \mathbf{Match}\ \overline{patc}}{\left(d\,\overline{a}\right) :: kcast\ \mathbf{Match}\ \left(d\,\overline{patc}\right) :: x_p}$$

$$\frac{}{.\ \mathbf{Match}\ .}$$

$$\frac{b\ \mathbf{Match}\ patc' \quad \overline{a}\ \mathbf{Match}\ \overline{patc}}{b\overline{a}\ \mathbf{Match}\ patc'\overline{patc}}$$

substitution abbreviation

$$-\left[\left(d\,\overline{patc}\right) :: x_p := d\,\overline{a}\right] = -\left[\overline{patc} := \overline{a}\right]\left[x_p := refl\right]$$

$$-\left[\left(d\,\overline{patc}\right) :: x_p := \left(d\,\overline{a}\right) :: kcast\right] = -\left[\overline{patc} := \overline{a}\right]\left[x_p := kcast\right]$$

Figure 8: Cast Language Matching and sub

Conjecture 7. *substitution of typed path preserves type*
equivalently the following rule is admissible

$$\frac{HK \vdash p : a \approx a' \quad x_p : a \approx a' \in H \quad HK \vdash b : B}{H\left[x_p := p\right]K \vdash b\left[x_p := p\right] : B\left[x_p := p\right]}$$

Conjecture 8. *substitution of kcasts preserve cast*
equivalently the following rule is admissible

$$\frac{HK \vdash kcast_K : a \approx a' \quad x_p : a \approx a' \in H \quad HK \vdash b : B}{H\left[x_p := kcast_K\right]K \vdash b\left[x_p := kcast\right] : B\left[x_p := kcast\right]}$$

Conjecture 9. *substitution of cast terms preserves path endpoints*
equivalently the following rule is admissible

$$\frac{HK \vdash a : A \quad x : A \in H \quad HK \vdash p : b \approx b'}{H\left[x := a\right]K \vdash p\left[x := a\right] : b\left[x := a\right] \approx b'\left[x := a\right]}$$

Conjecture 10. *substitution of typed paths preserves path endpoints*
equivalently the following rule is admissible

$$\frac{HK \vdash p : a \approx a' \quad x_p : a \approx a' \in H \quad HK \vdash q : b \approx b'}{H\left[x_p := p\right]K \vdash q\left[x_p := p\right] : b\left[x_p := p\right] \approx b'\left[x_p := p\right]}$$

Conjecture 11. *substitution of kcasts preserve cast*
equivalently the following rule is admissible

$$\frac{HK \vdash kcast_K : a \approx a' \quad x_p : a \approx a' \in H \quad HK \vdash b : B}{H\left[x_p := kcast_K\right]K \vdash q\left[x_p := kcast_K\right] : b\left[x_p := kcast\right] \approx b'\left[x_p := kcast\right]}$$

Finally we will conjecture the cast soundness.

$$\frac{x_p : A \approx A' \in H}{HK \vdash x_p : A \approx A'}$$

$$\frac{HK, k = \mathit{left} \vdash a : A \quad HK, k = \mathit{right} \vdash a : A'}{HK \vdash \mathit{Assert}_{k \Rightarrow a} : \lfloor a \rfloor_{k=\mathit{left}} \approx \lfloor a \rfloor_{k=\mathit{right}}}$$

does A=A'?

$$\frac{HK \vdash p : A \approx B \quad HK \vdash q : B \approx C}{HK \vdash p \circ q : A \approx C}$$

$$\frac{HK \vdash p : b \approx b \quad H, K \vdash b : B \quad H, K \vdash b' : B \quad H, x : B, K \vdash A}{HK \vdash \mathit{cong}_{x \Rightarrow a} p : a\,[x \coloneqq b] \approx a\,[x \coloneqq b']}$$

$$\frac{HK \vdash p : A \approx B}{HK \vdash p^{-1} : B \approx A}$$

$$\frac{HK \vdash p : (D\,\overline{a}) :: \mathit{kcast} \;\approx\; \left(D\,\overline{b}\right) :: \mathit{kcast}'}{HK \vdash \mathit{inTC}_i\, p : a_i \approx b_i}$$

and fully applied!

$$\frac{HK \vdash p : (d\,\overline{a}) :: \mathit{kcast} \;\approx\; \left(d\,\overline{b}\right) :: \mathit{kcast}'}{HK \vdash \mathit{inC}_i\, p : a_i \approx b_i}$$

and fully applied!

possibly force a matching type? but then it is unclear what type the conclusion should have

$$\frac{a' \equiv a \quad b' \equiv b \quad HK \vdash p : a \approx b}{HK \vdash p : a' \approx b'}$$

will need to adjust this if moves to a typed conversion rule

$$\frac{k = \mathit{left} \in K \quad HK \vdash a : A}{HK \vdash \{a \sim_{k,o,\ell} b\} : A}$$

$$\frac{k = \mathit{right} \in K \quad HK \vdash b : B}{HK \vdash \{a \sim_{k,o,\ell} b\} : B}$$

$$\frac{HK \vdash a : A \quad HK \vdash \mathit{kcast}_K : A \approx B}{HK \vdash a :: \mathit{kcast} \; : B}$$

$$\frac{\begin{array}{c} HK \vdash \overline{a} : \Delta \\ H, \Delta, K \vdash B : \star \\ \forall\, i\ \left(HK \vdash \overline{\mathit{pat}_i} : \Delta \quad H, \left(\overline{\mathit{pat}_i} : \Delta\right) K \vdash b_i : B\right) \\ \forall\, j\ \left(HK \vdash \overline{\mathit{pat}_j} : \Delta\right) \\ HK \vdash \overline{\overline{\mathit{patc}}}\,\overline{\overline{\mathit{patc}'}} : \Delta\ \textbf{complete} \end{array}}{\begin{array}{c} HK \vdash \mathsf{case}\,\overline{a}, \left\{\overline{|\,\mathit{patc}_i \Rightarrow b_i}\,\overline{|\,\mathit{patc}'_j \Rightarrow !_\ell}\right\} \\ : M\,[\Delta \coloneqq \overline{a}] \end{array}}$$

pattern expansion and pattern on context may need further exposition

$$\frac{\begin{array}{c} HK \vdash C : \star \\ HK \vdash \mathit{kcast}_K : a \approx a' \\ HK \vdash a : A \quad HK \vdash a' : A \\ \textbf{Head}(a) \neq \textbf{Head}(a') \end{array}}{HK \vdash !_{\mathit{kcast}} \; : C}$$

REMOVE TYPE RESTRICTION? the extra type restrictions is intended to force blame to the type level when needed, though this will not (cannot?) be invariant over reduction of paths in general

Figure 9: Cast Language typing and pathing rules

Conjecture 12. *The cast system preserves types and path endpoints over normalization*

Conjecture 13. *a well typed path in an empty context is a value, takes a step, or produces blame*

Conjecture 14. *A well typed term in an empty context is a value, takes a step, or produces blame*

19 Elaborating Eliminations

To make the overall system behave as expected we do not want to expose users to equality patterns, or force them to manually do the path bookkeeping. To work around this we extend a standard unification algorithm to cast patterns with instrumentation to remember paths that were required for the solution. Then if pattern matching is satisfiable, compile additional casts into the branch based on its assignments. Unlisted patterns can be checked to confirm they are unsatisfiable. If the pattern is unsatisfiable then elaboration can use the proof of unsatisfiability to construct explicit blame. If an unlisted pattern cannot be proven "unreachable" then we could warn the user, and like most functional programming languages, blame the incomplete match if that pattern ever occurs.

19.1 Preliminaries

As mentioned in the introduction we will need to add and remove justified casts from the endpoints of arguments. For instance, we will need to be able to generate $3 \approx x :: Nat$ from $3 \approx x$, $x : X$, and $X \approx Nat$. Fortunately the language is already expressive enough to embed these operations using an $Assert$ that does not bind a *same* assertion.

We will specify the shorthand $CastR_a p = Assert_{k \Rightarrow a::k=right,p} : a \approx a :: p$, similarly we can define $CastL$.

We can use use a similar construction to remove casts from an endpoint. Given a path $p : a :: q \approx b$, we can define the macro $UnCastR_a p = Assert_{k \Rightarrow a::k=right,q} \circ p \ : \ a \approx b$, similarly for $UnCastL$.

The surface language needs to be enriched with additional location metadata at each position where the two bidirectional typing modalities would cause a check in the surface language.

$$
\begin{array}{llll}
m... & ::= & ... & \\
 & | & \mathsf{case}\,\overline{n_\ell},\ \left\{\overline{|\,\overline{pat} \Rightarrow m_{\ell'}}\right\} & \text{data elim. without motive} \\
 & | & \mathsf{case}\,\overline{n_\ell},\ \langle \overline{x} \Rightarrow M_{\ell'} \rangle \left\{\overline{|\,\overline{pat} \Rightarrow m_{\ell''}}\right\} & \text{data elim. with motive}
\end{array}
$$

The implementation allows additional annotations along the motive, while this works within the bidirectional framework. The syntax is not presented here since the theory is already quite complicated.

where

19.2 Elaboration

"can", "could", weasel words until implementation is finalized

The biggest extension to the elaboration procedure in Chapter 3 is the path relevant unification and the insertion of casts to simulate surface language pattern matching. The unification and casting processes both work without k assumptions in scope, simplifying the possible terms that may appear.

The elaboration procedure uses the extended unification procedure to determine the implied type and assignment of each variable. In the match body casts are made so that variables behave as if they have the types and assignments consistent with the surface language. The original casting mechanism is still active, so it is possible that after all the casting types still don't line up. In this case primitive casts are still made at their given location.

add explicit rules for elaboration?

The elaboration algorithm is extremely careful to only add casts, this means erasure is preserved and evaluation will be consistent with the surface language.

Further the remaining properties from Chapter 3 probably still hold

Conjecture 15. *Every term well typed in the bidirectional surface language elaborates*

Conjecture 16. *Blame never points to something that checked in the bidirectional system*

$$\frac{}{U\left(\emptyset, \emptyset\right)}$$

$$\frac{U\left(E, u\right) \quad a \equiv a'}{U\left(\left\{p : a \approx a'\right\} \cup E, u\right)}$$

$$\frac{U\left(E\left[x \coloneqq a\right], u\left[x \coloneqq a\right]\right)}{U\left(\left\{p : x \approx a\right\} \cup E, u \cup \left\{p : x \approx a\right\}\right)}$$

actually a little incorrect, needs to use conq to concat the paths

$$\frac{U\left(E\left[x \coloneqq a\right], u\left[x \coloneqq a\right]\right)}{U\left(\left\{p : a \approx x\right\} \cup E, u \cup \left\{p^{-1} : x \approx a\right\}\right)}$$

$$\frac{U\left(\left\{p : a \approx a'\right\} \cup E, u\right) \quad a \equiv d\bar{b} \quad a' \equiv d\overline{b'}}{U\left(\left\{Con_i p : b_i \approx b'_i\right\}_i \cup E, u\right)}$$

fully applied

$$\frac{U\left(\left\{p : a \approx a'\right\} \cup E, u\right) \quad a \equiv D\bar{b} \quad a' \equiv D\overline{b'}}{U\left(\left\{TCon_i p : b_i \approx b'_i\right\}_i \cup E, u\right)}$$

fully applied

$$\frac{U\left(\left\{p : a :: q \approx a'\right\} \cup E, u\right)}{U\left(\left\{uncastLp : a \approx a\right\}_i \cup E, u\right)}$$

$$\frac{U\left(\left\{p : a \approx a' :: q\right\} \cup E, u\right)}{U\left(\left\{uncastRp : a \approx a'\right\} \cup E, u\right)}$$

fully applied

break cycle, make sure x is assignable

double check constraint order

correct vars in 4a

Figure 10: Surface Language Unification

20 Discussion and Future Work

20.1 Blame is not tight

Though the meta theory in this section is plausible, there are some awkward allowed behaviors. Blame may not be able to zero in as precisely as it seems is possible, when an assumption interacts with itself. For instance take the term under assumption k,

$\{\lambda x \Rightarrow x \sim_{k,o,\ell} \lambda x \Rightarrow x\}\,\{1 \sim_{k,o,\ell} 2\} \;\rightsquigarrow_*\; \left(\lambda x \Rightarrow \left\{x \sim_{k,o.app[x],\ell} x\right\}\right)\{1 \sim_{k,o,\ell} 2\} \;\rightsquigarrow_*\; \left\{2 \sim_{k,o.app[\{1 \sim_{k,o,\ell} 2\}],\ell} 0\right\}$
which will mistakenly give blame to the function when it is more reasonable to blame the argument. This situations is more complicated if we want to avoid blame when the two sides are mutually consistent $\{\lambda x \Rightarrow x \sim_{k,o,\ell} \lambda x \Rightarrow Not\ x\}\,\{true \sim_{k,o,\ell} f$ $\left(\lambda x \Rightarrow \left\{x \sim_{k,o.app[x],\ell} Not\ x\right\}\right)\{true \sim_{k,o,\ell} false\} \rightsquigarrow_* \left\{true \sim_{k,o.app[\{true \sim_{k,o,\ell} false\}],\ell} true\right\}$.

20.2 Types invariance along paths

It turns out that the system defined in Chapter 3 had the advantage of only dealing with equalities in the type universe. Extending to equalities over arbitrary type has vastly increased the complexity of the system. To make the system work paths are untyped, which seems inelegant. There is nothing currently preventing blame across type. For instance,

$\{1 \sim_{k,o,\ell} false\}$ will generate blame $1 \neq false$. While blame of $Nat \neq Bool$ will certainly result in a better error message. Several attempts were made to encode the type into the type assumption, but the resulting systems quickly became too complicated to work with. Some vestigial typing constraints are still in the system (such as on the explicit blame) to encourage this cleaner blame.

20.3 Elaboration is non-deterministic with regard to blame

Consider the case

```
case x <_:Id Nat 2 2 => S 2> {
| refl _ a => s a
}
```

that can elaborate to

```
case x <_:Id Nat 2 2 => S 2> {
| (refl A a)::p => (s (a::TCon0(p)) :: Cong uncastL(TCon1(p)))
}
```

where $p : Id\,A\,a\,a \approx Id\,Nat\,2\,2$, where $TCon_1 p$ selects the first position $p : Id\,A\,\underline{a}\,a \approx Id\,Nat\,\underline{2}\,2$. But this could also have elaborated to

```
case x <_:Id Nat 2 2 => S 2> {
| (refl A a)::p => (s (a::TCon0(p)) :: Cong uncastL(TCon2(p)))
}
```

relying on $p : Id\,A\,a\,\underline{a} \approx Id\,Nat\,2\,\underline{2}$. This can make a difference if the scrutinee is
$refl\ Nat\,2 :: Id\,Nat\,3\,2 :: Id\,Nat\,2\,2$
in one case blame will be triggered, in the other it will not. In this case it is possible to mix the blame from both positions, though this does not seem to extend in general since the consequences of inequality are undecidable in general and we intend to allow running programs if they can maintain their intended types.

20.4 Extending to Call-by-Value

As in Chapter 3, the system presented here does the minimal amount of checking to maintain type safety. This can lead to unexpected results, for instance consider the surface term

```
case (refl Nat 7 :: Id Nat 2 2) <_ => Nat> {
| refl _ a => a
}
```

This will elaborate into

```
case (refl Nat 7 :: Id Nat 2 2) <_ => Nat> {
| (refl A a)::p => a::TCon0(p)
}
```

which will evaluate to $7 :: \texttt{Nat}$ without generating blame. And indeed we only ever asserted that the result was of type `Nat`.

In the implementation, some of this behavior is avoided by requiring type arguments in a cast be run call-by-value. This restriction will blame $7 \neq 2$ before the cast is even evaluated. expand

20.5 Efficiency

The system defined here is brutally inefficient.

In theory the system has an arbitrary slow down. As in Chapter 3, a cast that relies on non-terminating code can itself cause additional non-termination as paths are resolved.

As written there are many redundant computations, and trying every combination of assumptions is very inefficient. Currently the implementation is quite slow, though there are several ways to speed things up. Caching redundant computation would help. Having a smarter embedding of k assignments would remove redundant work directly. To some extent *Cong* and *Assert* can be made multi-arity to allow fewer jumps. But most helpful of all would be simplifying away unneeded casts. More advanced options include using proof search to find casts that will never cause an error.

paremetricity

relation to fun-ext

warnings

20.6 Relation to UIP

Pattern matching as outlined in the last Chapter (which follows from [Coq92]) implies the **uniqueness of identity proofs** (UIP)[4]. UIP states that every proof of identity is equal to refl (and thus unique), and is not provable in many type theories. In univalent type theories UIP is directly contradicted by the "non-trivial" equalities, required to equate isomorphisms and Id. UIP is derivable in the surface language by following pattern match

```
case x <pr : Id A a a => Id (Id A a a) pr (refl A a) > {
| refl A a => refl (Id A a a) (refl A a)
}
```

This type checks since unification will assign $pr := refl\,A\,a$ and under that assumption $refl\ (Id\ A\ a\ a)\ (refl\ A\ a) : Id\ (Id\ A\ a\ a)\ (refl\ A\ a)\ (refl\ A\ a)$. Like univalent type theories, the cast language has its own nontrivial equalities, so it might seem that the cast language would also contradict UIP . But it is perfectly compatible, and will elaborate. One interpretation is that though there are multiple "proofs" of identity, we don't care which one is used.

interpreta… aways?

20.7 Future work

Make equalities visible in the surface syntax

The system here has some simple inspiration that could be extended into pattern matching syntax more generally. It seems useful to be able to read equational information out of patterns, especially in settings with rich treatment of equality. Matching equalities directly could be a semi-useful feature in Agda, or in univalent type theories such as CTT.

21 Related work

This work was previously presented as an extended abstract at the TyDE workshop

cite

, the version there reflected a less plausible meta-theory based on earlier implementation experiments.

CTT data is related?

[4] Also called **axiom k**

References

[Coq92] Thierry Coquand. Pattern matching with dependent types. In *Proceedings of the Workshop on Types for Proofs and Programs*, pages 71–83. Citeseer, 1992.

Part III
TODO

check by value

Todo list

22 notes

there are several simpler systems that can be worked through: eliminator style patterns, cast patterns, but to bring it all together we need congruence over functions.

adding paths and path variables means that constructs can still fail at runtime, but they can blame the actually problematic components

validating the K axiom, not that equalities are unique, merely that we don't care which one of the unique equalities is used.

23 unused

```
case x <pr : Id A a a => Id (Id A a a) pr (refl A a) > {
| (refl A' a') :: p =>
refl (((Id A')::(A -> A -> *)) (a'::A) (a'::A) ) :: (pr' : (Id A a a) -> Id (Id A a a) pr' pr')
(refl A')::((a : A) -> Id A a a) (a'::A)) ::

}
```

Where $p : Id\,A'\,a'\,a' \approx Id\,A\,a\,a$, ...

...

$$\frac{H\,K\ \mathbf{ok}}{HK \vdash \lozenge : .}\ \ldots$$

$$\frac{H, x : A; K \vdash \Delta \quad H; K \vdash A : \star \quad H; K \vdash patc : \Delta}{HK \vdash x, patc\ :\ (x : A)\,\Delta}\ \ldots$$

$$\frac{\begin{array}{c} d\ :\ \Theta \to D\overline{b} \in H \\ HK \vdash \overline{patc'} : \Theta \\ H, \left(\overline{patc'} : \Theta\right), x_p : D\overline{b} \approx D\overline{a}, K \vdash patc : \Delta\left[x \coloneqq d\,\overline{patc'} ::_{x_p}\right] \end{array}}{HK \vdash d\,\overline{patc'} ::_{x_p}, patc\ :\ (x : D\overline{a})\,, \Delta}\ \ldots$$

...

$$\frac{HK \vdash A : \star}{HK \vdash x : A}\ \ldots$$

$$\frac{\Gamma, x : M \vdash \Delta \quad \Gamma \vdash m : M \quad \Gamma \vdash \overline{n}\,[x \coloneqq m] : \Delta\,[x \coloneqq m]}{\Gamma \vdash m, \overline{n}\ :\ x : M, \Delta}\ \ldots$$

$$\frac{\Gamma\,\mathbf{ok} \quad \mathsf{data}\,D\,\Delta \in \Gamma}{\Gamma \vdash D\ :\ \Delta \to *}\ \ldots$$

```
-- standard data in normal form, 3
S (S (S 0))

-- cast data in normal form
S (S (S 0) :: Nat ) :: Nat :: Nat :: Nat
S (S (S 0) :: Nat ) :: Bool :: Nat
True :: Nat

-- cast pattern matching
case x <_ => Bool> {
| (Z :: _) => True
| (S (Z :: _) :: _) => True
| (S (S :: _) :: _) => False
}

-- extract specific blame,
-- c is a path from Bool~Nat
case x <_ => Nat> {
| (S ((true::c)::_) :: _) =>
 add (false :: c) 2
}

-- can reconstitute any term,
-- not always possible with unification
-- based pattern matching
case x <_:Nat => Nat> {
| (Z :: c) => Z :: c
| (S x :: c) => S x :: c
}

-- direct blame
case x <_ => Nat> {
| (S (true::c) :: _) => Bool =/=c Nat
}

peek x =
case x <_: Id Nat 0 1 => Nat> {
  | (refl x :: _) => x
}

peek (refl 4 :: Id Nat 0 1) = 4
```

to stylize consistently, should use math font, or like a nice image

break into smaller more relevant examples

Figure 11: Cast Pattern Matching

$$\frac{\Gamma\,\mathbf{ok} \quad d : \Theta \to D\overline{m} \in \Gamma}{\Gamma \vdash d : \Theta \to D\overline{m}} \ldots$$

...

$$\frac{}{HK \vdash x : A} \ldots$$

$$\frac{H \vdash A : \star}{H \vdash refl : A \approx A}$$

$$\frac{H \vdash B : \star \quad H, x : B \vdash C : \star \quad H \vdash b : B \quad H \vdash b' : B \quad C\,[x \coloneqq b] \equiv A \quad C\,[x \coloneqq b'] \equiv A'}{H \vdash A_{\ell.x \Rightarrow C} A' : A \approx A'}$$

ALT, would then need to resolve endpoint def equality

$$\frac{H \vdash a : A \quad H \vdash a' : A \quad H, x : A \vdash C : \star}{H \vdash assert_{\ell.(a=a':A).x \Rightarrow C} : C\,[x \coloneqq a] \approx C\,[x \coloneqq a']}$$

$$\frac{H \vdash p : A \approx B \quad H \vdash p' : B \approx C}{H \vdash p\,p' : A \approx C}$$

$$\frac{H \vdash p : A \approx B}{H \vdash rev\,p : B \approx A}$$

typing rules

$$\frac{H \vdash C : \star \quad H \vdash p : A \approx B \quad AandBDisagree}{H \vdash A \neq_p B : C}$$

$$\frac{H \vdash a : A \quad H, x : B \vdash C : \star \quad C\,[x \coloneqq b] \equiv A \quad C\,[x \coloneqq b'] \equiv B}{H \vdash a ::_{A,\ell.x \Rightarrow C} B}$$

ALT

$$\frac{H \vdash a : A \quad H \vdash a' : A \quad H, x : A \vdash C : \star \quad H \vdash a : c\,[x \coloneqq a]}{H \vdash c ::_{\ell(a=a':A).x \Rightarrow C} \quad : C\,[x \coloneqq a']}$$

ALT remove concrete casts and merely use a symbolic cast instead?
...

$$\frac{H \vdash a : A \quad H, x : B \vdash C : \star \quad C\,[x \coloneqq b] \equiv A \quad C\,[x \coloneqq b'] \equiv B \quad p : b \approx b'}{H \vdash a ::_{A,p.x \Rightarrow C} B}$$

ALT

$$\frac{H \vdash c : C\,[x \coloneqq a] \quad H, x : A \vdash C : \star \quad H \vdash p : a \approx a'}{H \vdash c ::_{p.x \Rightarrow C} \quad : C\,[x \coloneqq a']}$$

$$\frac{\begin{array}{c} H \vdash \overline{a} : \Delta \\ H, \Delta \vdash B : \star \\ \forall\, i\, \left(H \vdash Gen\left(\overline{patc}_i : \Delta, \Theta\right) \quad \Gamma, \Theta \vdash m : M\left[\Delta \coloneqq \overline{patc}_i\right]\right) \\ H \vdash \overline{\overline{patc}} : \Delta\ \mathbf{complete} \end{array}}{\begin{array}{c} \mathsf{case}\,\overline{a},\, \langle \Delta \Rightarrow B \rangle \left\{\overline{\mid \overline{patc} \Rightarrow b}\right\} \\ : M\,[\Delta \coloneqq \overline{n}] \end{array}} \ldots$$

Gen is defined as

$$\frac{}{H \vdash Gen\,(. : ., .)} \ldots$$

$$\frac{\sim H \vdash A : \star \sim}{H \vdash Gen\,(x : (x : A),\ x : A)} \ldots$$

$$\frac{\sim H\vdash A:\star\sim}{H\vdash Gen\left(x:A,\ x:A\right)}\ldots$$

$$\frac{d\ :\ \Theta\to D\overline{a}\in H\quad H\vdash Gen\left(\overline{pat_c}:\Theta,\Delta\right)}{H\vdash Gen\left(d\overline{pat_c}::_{x_p}:D\overline{b},\ \Delta,x_p:D\overline{a}\approx D\overline{b}\right)}\ldots$$

$$\frac{H\vdash Gen\left(pat_c:A,\Theta\right)\quad H,\Theta\vdash Gen\left(\overline{pat_c}:\Delta\left[x\coloneqq pat_c\right],\Theta'\right)}{H\vdash Gen\left(pat_c\overline{pat_c}:\left(x:A,\Delta\right),\Theta\Theta'\right)}\ldots$$

other rules similar to the surface lang

observations,

o	::=	...	
	|	$o.App[a]$	application
	|	$o.TCon[i]$	type cons. arg.
	|	$o.DCon[i]$	data cons. arg.

old style red rules

$$\overline{rev\ (p\,p')\rightsquigarrow(rev\,p')\,(rev\,p)}$$

$$\overline{inTC_i\ (p\,p')\rightsquigarrow(inTC_i\,p')\,(inTC_i\,p)}$$

$$\overline{inC_i\ (p\,p')\rightsquigarrow(inC_i\,p')\,(inC_i\,p)}$$

$$\overline{inTC_i\,refl\rightsquigarrow refl}$$

$$\overline{inC_i\,refl\rightsquigarrow refl}$$

$$\frac{\overline{a}_i=a'\ \overline{c}_i=c'\ \overline{b}_i=b'}{inTC_i\ \left(D\,\overline{a}_{\ell.D\,\overline{c}}D\,\overline{b}\right)\rightsquigarrow a'_{\ell.c'}b'}$$

$$\overline{inC_i\ ((a::A)_{\ell.c}\,b)\rightsquigarrow inC_i\ (a_{\ell.c}b)}$$

$$\overline{inC_i\ (a_{\ell.c}\,(b::B))\rightsquigarrow inC_i\ (a_{\ell.c}b)}$$

$$\overline{inC_i\ \left(a_{\ell.(c::C)}b\right)\rightsquigarrow inC_i\ (a_{\ell.c}b)}$$

$$\frac{\overline{a}_i=a'\ \overline{c}_i=c'\ \overline{b}_i=b'}{inTC_i\ \left(d\,\overline{a}_{\ell.d\,\overline{c}}d\,\overline{b}\right)\rightsquigarrow a'_{\ell.c'}b'}$$

$$\overline{a::_{A,p\,refl,x.C}\ B\rightsquigarrow a::_{A,p,x.C}\ B}$$

$$\frac{}{\begin{array}{c}a::_{A,p\,A'_{\ell.C''}B',x.C}\ B\rightsquigarrow\\ a::_{A,p,x.C}\ C\left[x\coloneqq A'\right]::_{\ell.C[x\coloneqq C'']}\ C\left[x\coloneqq B'\right]\end{array}}\ c$$

c?

$$\overline{(a::_{A,p,x.C}\ C)\sim_{\ell o}b\rightsquigarrow a\sim_{\ell o}b}$$

$$\overline{a\sim_{\ell o}(b::_{B,p,x.C}\ C)\rightsquigarrow a\sim_{\ell o}b}$$

...

path var,
x_p
assertion index,
k
assertion assumption,

kin	::=	$k = left \mid k = right$	

casts under assumption,

$kcast$	::=	$\overline{\overline{kin},p;}$	

path exp.,

p, p'	::=	x_p	
	$\mid$	$Assert_{k\Rightarrow C}$	concrete cast
	$\mid$	$refl$	
	$\mid$	pp'	
	$\mid$	p^{-1}	
	$\mid$	$inTC_i\, p$	
	$\mid$	$inC_i\, p$	
	$\mid$	$uncastL_{kcast}\, p$	
	$\mid$	$uncastR_{kcast}\, p$	

cast pattern,

$patc$	::=	$x \mid d\,\overline{patc} ::_{x_p}$	

cast expression,

$a...$	::=	...	
	$\mid$	D	type cons.
	$\mid$	d	data cons.
	$\mid$	$\mathsf{case}\,\overline{a},\left\{\overline{\mid \overline{patc} \Rightarrow b}\,\overline{\mid \overline{patc'} \Rightarrow !_\ell}\right\}$	data elim.
	$\mid$	$!_p$	force blame
	$\mid$	$a :: kcast$	cast
	$\mid$	$\{a \sim_{k,o,\ell} b\}$	assert same

observations,

o	::=	...	
	$\mid$	$o.App[a]$	application
	$\mid$	$o.TCon[i]$	type cons. arg.
	$\mid$	$o.DCon[i]$	data cons. arg.

$$\frac{C \rightsquigarrow C'}{Assert_{k\Rightarrow C} \rightsquigarrow Assert_{k\Rightarrow C'}}$$

$$\frac{}{refl\, p \rightsquigarrow p}$$

$$\frac{}{p\, refl \rightsquigarrow p}$$

$$\frac{}{(q\, p)^{-1} \rightsquigarrow p^{-1}\, q^{-1}}$$

$$\frac{q \rightsquigarrow q' \quad p}{q\, p \rightsquigarrow q'\, p}$$

$$\frac{q\ \mathbf{Val} \quad p \rightsquigarrow p'}{q\, p \rightsquigarrow q\, p'}$$

$$\frac{}{(Assert_{k\Rightarrow C})^{-1} \rightsquigarrow Assert_{k\Rightarrow \mathbf{Swap}_k C}}$$

$$\frac{}{inTC_i\,(Assert_{k\Rightarrow D\overline{A}}) \rightsquigarrow Assert_{k\Rightarrow A_i}}$$

$$\frac{}{inC_i\,(Assert_{k\Rightarrow d\overline{A}}) \rightsquigarrow Assert_{k\Rightarrow A_i}}$$

TODO review this

$$\frac{remove\, k = left\, casts \quad a\ \mathbf{whnf}}{uncastL\left(Assert_{k\Rightarrow a::\overline{\overline{kin,p;}}}\right) \rightsquigarrow Assert_{k\Rightarrow a::\overline{\overline{kin',p';}}}}$$

probly need to modify substution

$$\frac{}{refl^{-1} \rightsquigarrow refl}$$

$$\frac{}{inTC_i\,(refl) \rightsquigarrow refl}$$

$$\frac{}{inC_i\,(refl) \rightsquigarrow refl}$$

TODO review this

$$\frac{}{uncastL\ (refl) \rightsquigarrow ?}$$

term redcutions

$$\frac{p \rightsquigarrow p'}{!_p \rightsquigarrow !_{p'}}$$

$$\frac{}{\left\{a :: \overline{\overline{kin},p;}\overline{kin},q\, Assert_{k\Rightarrow C}; \overline{\overline{kin'},p';} \sim_{k,o,\ell} b\right\} \rightsquigarrow \left\{a :: \overline{\overline{kin},p;}\overline{kin},q; \overline{\overline{kin'},p';} \sim_{k,o,\ell} b\right\} :: \overline{kin},k = left\, Assert_{k\Rightarrow C};}$$

symetric around $\sim$

$$\frac{}{\{\star \sim_{k,o,\ell} \star\} \rightsquigarrow \star}$$

$$\frac{}{\{(x:A) \to B \sim_{k,o,\ell} (x:A') \to B'\} \rightsquigarrow (x : \{A \sim_{k,o.arg,\ell} A'\}) \to \left\{B \sim_{k,o.bod[x],\ell} B'\right\}}$$

$$\frac{}{\{\mathsf{fun}\, f\, x \Rightarrow b \sim_{k,o,\ell} \mathsf{fun}\, f\, x \Rightarrow b'\} \rightsquigarrow \mathsf{fun}\, f\, x \Rightarrow \left\{b \sim_{k,o.app[x],\ell} b'\right\}}$$

$$\frac{}{\left\{d\overline{a} \sim_{k,o,\ell} d\overline{a'}\right\} \rightsquigarrow d\overline{\left\{a_i \sim_{k,o.o.DCon[i],\ell} a_i'\right\}}}$$

$$\frac{}{\left\{D\overline{a} \sim_{k,o,\ell} D\overline{a'}\right\} \rightsquigarrow D\overline{\overline{\left\{a_i \sim_{k,o.o.TCon[i],\ell} a_i'\right\}}}}$$

$$\frac{}{a :: \overline{\overline{kin,;}} \rightsquigarrow a}$$

$$\frac{pointwise\ concatination}{\left(a :: \overline{\overline{kin,p;}}\right) :: \overline{\overline{kin',p';}} \rightsquigarrow ...}$$

$$\frac{}{\begin{pmatrix} & ... & \\ a :: & kin,q\ Assert_{k\Rightarrow(x:A)\to B}; & \\ & ... & \end{pmatrix} b \rightsquigarrow \left(\begin{pmatrix} & ... & \\ a :: & kin,q\ Assert_{k\Rightarrow(x:A)\to B}; & \\ & ... & \end{pmatrix} \left(b :: kin, Assert_{k\Rightarrow \mathbf{Swap}_k A};\right)\right) :: kin, Assert_{k\Rightarrow B\left[x:=\right.}}$$

$$\frac{Match\,\overline{a}\,patc_i}{\mathsf{case}\,\overline{a},\left\{\overline{|\,\overline{patc_i \Rightarrow} b_i}\overline{|\,\overline{patc' \Rightarrow}!_\ell}\right\} \rightsquigarrow b_i\,[patc_i := \overline{a}]}$$

...

$$\frac{p\ \mathbf{Val}}{q \circ refl \circ p \rightsquigarrow q \circ p}$$

$$\frac{p\ \mathbf{Val} \quad q\ \mathbf{Val}}{(q \circ p)^{-1} \rightsquigarrow p^{-1} \circ q^{-1}}$$

$$\frac{q \rightsquigarrow q'}{p \circ q \rightsquigarrow p \circ q'}$$

$$\frac{q\ \mathbf{Val} \quad p \rightsquigarrow p'}{p \circ q \rightsquigarrow p' \circ q}$$