Hazel MetaPhi: 9-type-aliases

July 3, 2021

How to read

800000 kinds
008000 types (constructors)
000080 terms

Issues

Issue 1: Algorithmic rules are not "officially" algorithmic
Since we are going all out to Do It Right™, it should be noted that the declarative/algorithmic bifurcation is not complete with :: $(\Delta; \Phi \vdash \tau :: \kappa)$.
For example, kind analysis is premissed on $\lesssim$ $(\Delta; \Phi \vdash \kappa_1 \lesssim \kappa_2)$ at `KAASubsume`, which is itself premissed on $\equiv$ $(\Delta; \Phi \vdash \tau_1 \equiv \tau_2)$ at `KCRespectEquiv`, which is itself premissed on :: $(\Delta; \Phi \vdash \tau :: \kappa)$ at `KCESingEquiv`.
Explicitly algorithmic counterparts to $\lesssim$ and $\equiv$ need to be defined. Suggested notation: $\lessapprox$ and $\Longleftrightarrow$ (faintly reminiscent of Stone and Harper).

Issue 2: A $t :: \kappa \in \Phi \implies t :: \mathtt{S}(t)$ (like) rule is problematic
When $\forall \tau. \Delta; \Phi \vdash \tau :: \kappa \implies \Delta; \Phi \vdash \tau :: \mathtt{Type}$, as is now, this is fine (except for improperly (non)contexted holes $\forall \tau. \vdash \tau :: \mathtt{Type}$ I would think). But in the presence of non`Type` kinds, this rule no longer conveys the correct meaning (intuitively speaking). For example, if we redefined $\mathtt{list}(\tau)$ to be a type constructor proper instead of a builtin type schema, such that $\vdash \mathtt{list} :: \mathtt{Type} \to \mathtt{Type}$, and had nonstifled type aliases such that $t :: \mathtt{list}$, $\mathtt{S}(t)$ certainly does not mean what we want (I say nonstifled since we could technically skirt this issue by demanding type alias definitions to be a `Type`, but that would clearly be stifling– and no langauge does this for obvious reasons).
So the logical conclusion is higher order singletons. Thankfully, Stone and Harper showed that HOSingletons are definable in terms of "vanilla" singletons. Thus, we do not need to add HOSingletons to the object language itself– only to the metalangauge. HOSingletons should make the metatheory proofs more general, even without considering type constructors. The question is how HOSingletons and holes should interact.

Not Issues-- but worth mentioning

Nibwm 1: In the presence of higher order types/type constructors, there exists a more nuanced notion of type equivalence (extensionality) in which equivalence depends on the kind at which the types are compared

Without "real" subkinding, examples are a bit more contrived.

From Stone and Harper (2006):

$$\vdash \lambda t{::}\mathtt{Type}.t \overset{\mathtt{S(Int)}\to\mathtt{Type}}{\equiv} \lambda t{::}\mathtt{Type}.\mathtt{Int}$$

should be derivable but

$$\vdash \lambda t{::}\mathtt{Type}.t \overset{\mathtt{Type}\to\mathtt{Type}}{\equiv} \lambda t{::}\mathtt{Type}.\mathtt{Int}$$

should not, where $\vdash \mathtt{Type} \to \mathtt{Type} \lesssim \mathtt{S(Int)} \to \mathtt{Type}$

Interestingly

$$\vdash \lambda t{::}\mathtt{Type}.t \overset{\mathtt{S(Int)}\to\mathtt{S(Int)}}{\equiv} \lambda t{::}\mathtt{Type}.\mathtt{Int}$$

should also be derivable, where $\vdash \mathtt{S(Int)}{\to}\mathtt{S(Int)} \lesssim \mathtt{S(Int)}{\to}\mathtt{Type}$

With "real" subkinding, this behavior is more serious.

From Aspinall (1995), using singleton types and $\mathtt{Nat} \leq \mathtt{Int}$ ("real" subtyping):

$$\vdash \lambda \mathtt{x}{:}\mathtt{Int}.\mathtt{if}\ \mathtt{x} \geq 0\ \mathtt{then}\ \mathtt{x}\ \mathtt{else}\ 2 * \mathtt{x} \overset{\mathtt{Nat}\to\mathtt{Int}}{\equiv} \lambda \mathtt{x}{:}\mathtt{Int}.\mathtt{x}$$

should be derivable but

$$\vdash \lambda \mathtt{x}{:}\mathtt{Int}.\mathtt{if}\ \mathtt{x} \geq 0\ \mathtt{then}\ \mathtt{x}\ \mathtt{else}\ 2 * \mathtt{x} \overset{\mathtt{Int}\to\mathtt{Int}}{\equiv} \lambda \mathtt{x}{:}\mathtt{Int}.\mathtt{x}$$

should not.

I do not believe this more nuanced view of equality buys anything for us.

Attachments

9-type-aliases marked up with preliminary declarative statics notes

Attachment 1

single numbers   stone, harper 2006

43.X   PFPL

$$
\begin{array}{rcll}
\textsf{BinOp} & \oplus & ::= & \texttt{Product} \mid \texttt{Sum} \mid \texttt{Arrow} \\
\textsf{Kind} & \kappa & ::= & \texttt{Ty} \mid \texttt{KHole} \mid \texttt{S}(\tau) \\
\textsf{ConstantTypes} & c & ::= & \texttt{Int} \mid \texttt{Float} \mid \texttt{Bool} \\
\textsf{UserHTyp} & \hat{\tau} & ::= & c \mid \hat{\tau}_1 \oplus \hat{\tau}_2 \mid \texttt{list}(\hat{\tau}) \mid (\!|\,|\!)^u \mid (\!|\hat{\tau}|\!)^u \\
\textsf{InternalHTyp} & \tau & ::= & c \mid \tau_1 \oplus \tau_2 \mid \texttt{list}(\tau) \mid (\!|\,|\!)^u \mid (\!|\tau|\!)^u \quad (\!|t|\!)^u \\
\textsf{TypeVars} & t & & \\
\textsf{TypePattern} & \rho & ::= & t \mid (\!|\,|\!) \mid (\!|t|\!) \\
\textsf{UserExpression} & e & ::= & \texttt{type}\ \rho\ \texttt{=}\ \hat{\tau}\ \texttt{in}\ e \mid \textit{elided} \\
\textsf{InternalExpression} & \tau & ::= & \texttt{type}\ \rho\ \texttt{=}\ \tau : \kappa\ \texttt{in}\ d \mid \textit{elided}
\end{array}
$$

$\boxed{\Delta; \Phi \vdash \kappa_1 \lesssim \kappa_2}$ κ_1 is a consistent subkind of κ_2

$$
\texttt{KCHoleL}\ \frac{}{\Delta; \Phi \vdash \texttt{KHole} \lesssim \kappa}
\qquad
\texttt{KCHoleR}\ \frac{}{\Delta; \Phi \vdash \kappa \lesssim \texttt{KHole}}
\qquad
\texttt{KCRespectEquiv}\ \frac{\Delta; \Phi \vdash \kappa_1 \equiv \kappa_2}{\Delta; \Phi \vdash \kappa_1 \lesssim \kappa_2}
$$

$$
\texttt{KCSubsumption}\ \frac{\Delta; \Phi \vdash \tau \Leftarrow \texttt{Ty}}{\Delta; \Phi \vdash \texttt{S}(\tau) \lesssim \texttt{Ty}} \quad 7
$$

$\boxed{t\ \texttt{valid}}$ t is a valid type variable

t is valid if it is not a builtin-type or keyword, begins with an alpha char or underscore, and only contains alphanumeric characters, underscores, and primes.

$\boxed{\Delta; \Phi \vdash \kappa\ \texttt{kind}}$ κ forms a kind

$$
\texttt{KFTy}\ \frac{}{\Delta; \Phi \vdash \texttt{Ty kind}}
\qquad
\texttt{KFHole}\ \frac{}{\Delta; \Phi \vdash \texttt{KHole kind}}
\qquad
\texttt{KFSing}\ \frac{\Delta; \Phi \vdash \tau \Leftarrow \texttt{Ty}}{\Delta; \Phi \vdash \texttt{S}(\tau)\ \texttt{kind}} \quad 43.2a
$$

Attachment 1

$\boxed{\Delta; \Phi \vdash \kappa_1 \equiv \kappa_2}$ κ_1 is equivalent to κ_2

KERefl
$$\frac{}{\Delta; \Phi \vdash \kappa \equiv \kappa}$$

KESymm
$$\frac{\Delta; \Phi \vdash \kappa_1 \equiv \kappa_2}{\Delta; \Phi \vdash \kappa_2 \equiv \kappa_1}$$

KETrans
$$\frac{\Delta; \Phi \vdash \kappa_1 \equiv \kappa_2 \qquad \Delta; \Phi \vdash \kappa_2 \equiv \kappa_3}{\Delta; \Phi \vdash \kappa_1 \equiv \kappa_3}$$

KESingEquiv
$$\frac{\Delta; \Phi \vdash \tau_1 \equiv \tau_2}{\Delta; \Phi \vdash \mathtt{S}(\tau_1) \equiv \mathtt{S}(\tau_2)}$$

would need to define labelled singletons for KHole

$\boxed{\Delta; \Phi \vdash \tau \Rightarrow \kappa}$ τ synthesizes kind κ

p.687 St, Harp 2006

KSConst
$$\frac{}{\Delta; \Phi \vdash c \Rightarrow \mathtt{S}(c)}$$

KSVar
$$\frac{t : \kappa \in \Phi}{\Delta; \Phi \vdash t \Rightarrow \mathtt{S}_{\kappa}(t)}$$

KSUVar
$$\frac{t \notin \mathsf{dom}(\Phi) \qquad u :: \kappa \in \Delta}{\Delta; \Phi \vdash (\!| t |\!)^u \Rightarrow \cancel{\mathtt{KHole}}\ \kappa}$$

KSBinOp
$$\frac{\Delta; \Phi \vdash \tau_1 \Leftarrow \cancel{\mathtt{S}(\tau_1)}\ \mathrm{Ty} \qquad \Delta; \Phi \vdash \tau_2 \Leftarrow \cancel{\mathtt{S}(\tau_2)}\ \mathrm{Ty}}{\Delta; \Phi \vdash \tau_1 \oplus \tau_2 \Rightarrow \mathtt{S}(\tau_1 \oplus \tau_2)}$$

KSList
$$\frac{\Delta; \Phi \vdash \tau \Leftarrow \cancel{\mathtt{S}(\tau)}\ \mathrm{Ty}}{\Delta; \Phi \vdash \mathtt{list}(\tau) \Rightarrow \mathtt{S}(\mathtt{list}(\tau))}$$

KSEHole
$$\frac{u :: \kappa \in \Delta}{\Delta; \Phi \vdash (\!|\,|\!)^u \Rightarrow \kappa}$$

KSNEHole
$$\frac{u :: \kappa \in \Delta \qquad \Delta; \Phi \vdash \tau \Rightarrow \kappa'}{\Delta; \Phi \vdash (\!| \tau |\!)^u \Rightarrow \kappa}$$

$\boxed{\Delta; \Phi \vdash \tau \Leftarrow \kappa}$ τ analyzes against kind κ

KAASubsume
$$\frac{\Phi \vdash \tau \Rightarrow \kappa' \qquad \Delta; \Phi \vdash \kappa' \lesssim \kappa}{\Delta; \Phi \vdash \tau \Leftarrow \kappa}$$

KAVar
$$\frac{t :: \kappa_1 \in \Phi \qquad \kappa_1 \lesssim \kappa}{\Delta; \Phi \vdash t \Leftarrow \kappa}$$

Attachment 1

$\boxed{\Delta; \Phi \vdash \tau_1 \equiv \tau_2}$ τ_1 is equivalent to τ_2

KCESymm
$$\frac{\Delta; \Phi \vdash \tau_1 \equiv \tau_2}{\Delta; \Phi \vdash \tau_2 \equiv \tau_1}$$

KCETrans
$$\frac{\Delta; \Phi \vdash \tau_1 \equiv \tau_2 \qquad \Delta; \Phi \vdash \tau_2 \equiv \tau_3}{\Delta; \Phi \vdash \tau_1 \equiv \tau_3}$$

KCESingEquiv
$$\frac{\Delta; \Phi \vdash \tau_1 \Leftarrow \mathtt{S}_{\kappa}(\tau_2)}{\Delta; \Phi \vdash \tau_1 \equiv \tau_2}$$

KCERefl
$$\boxed{\frac{\tau :: \kappa}{\Delta; \Phi \vdash \tau \equiv \tau}}$$

~~KCEConst~~
$$\cancel{\frac{}{\Delta; \Phi \vdash c \equiv c}}$$

~~KCEVar~~
$$\cancel{\frac{t : \kappa \in \Phi}{\Delta; \Phi \vdash t \equiv t}}$$

KCEBinOp
$$\frac{\Delta; \Phi \vdash \tau_1 \equiv \tau_2 \qquad \Delta; \Phi \vdash \tau_3 \equiv \tau_4}{\Delta; \Phi \vdash \tau_1 \oplus \tau_3 \equiv \tau_2 \oplus \tau_4}$$

KCEList
$$\frac{\Delta; \Phi \vdash \tau_1 \equiv \tau_2}{\Delta; \Phi \vdash \mathtt{list}(\tau_1) \equiv \mathtt{list}(\tau_2)}$$

KCEEHole
$$\boxed{\frac{u :: \kappa \in \Delta}{\Delta; \Phi \vdash ⦇⦈^u \equiv ⦇⦈^u}}$$

* if $u :: \kappa \notin \Delta$ KCERefl says OK

KCENEHole
$$\boxed{\frac{u :: \kappa \in \Delta \qquad \Delta; \Phi \vdash \tau \Leftarrow \kappa'}{\Delta; \Phi \vdash ⦇\tau⦈^u \equiv ⦇\tau⦈^u}}$$

* sim

$\boxed{\Delta; \Phi \vdash \tau :: \kappa}$ τ is well formed at kind κ

$$\frac{t :: \kappa \in \Phi}{\Delta; \Phi \vdash t :: \kappa} \; 17$$

$$\frac{\Delta; \Phi \vdash \tau :: Ty}{\Delta; \Phi \vdash \mathtt{list}(\tau) :: Ty}$$

$$\frac{\Delta; \Phi \vdash \tau :: \kappa_1 \qquad \Delta; \Phi \vdash \kappa_1 \lesssim \kappa}{\Delta; \Phi \vdash \tau :: \kappa} \; 43.2d$$

$$\boxed{\frac{\Delta; \Phi \vdash \tau :: Ty}{\Delta; \Phi \vdash \tau :: S(\tau)} \; \begin{matrix}23\\43.2b\end{matrix}}$$

$$\frac{u :: \kappa \in \Delta}{\Delta; \Phi \vdash ⦇⦈^u :: \kappa}$$

$$\boxed{\frac{\Delta; \Phi \vdash \tau :: \kappa}{\Delta; \Phi \vdash \tau :: S_{\kappa}(\tau)} \; \text{replaces 23?}}$$

$$\frac{}{\Delta; \Phi \vdash c :: Ty} \; 16$$

$$\frac{\Delta; \Phi \vdash u :: \kappa \in \Delta \qquad \Delta; \Phi \vdash \tau :: \kappa_1}{\Delta; \Phi \vdash ⦇\tau⦈^u :: \kappa}$$

$$\frac{t \notin dom(\Phi) \qquad u :: \kappa \in \Delta}{\Delta; \Phi \vdash ⦇t⦈^u :: \kappa}$$

$$\frac{\Delta; \Phi \vdash \tau_1 :: Ty \qquad \Delta; \Phi \vdash \tau_2 :: Ty}{\Delta; \Phi \vdash \tau_1 \oplus \tau_2 :: Ty}$$

$S_{Ty}(\tau) := S(\tau)$

$S_{S(\tau_1)}(\tau) := S(\tau)$

$\boxed{S_{KHole}(\tau) := KHole}$

Attachment 1

$\boxed{\Phi \vdash \hat{\tau} \Rightarrow \kappa \rightsquigarrow \tau \dashv \Delta}$ $\hat{\tau}$ synthesizes kind κ and elaborates to τ

$$\texttt{TElabSConst}\quad \frac{}{\Phi \vdash c \Rightarrow \texttt{S}(c) \rightsquigarrow c \dashv \cdot}$$

$$\texttt{TElabSBinOp}\quad \frac{\Phi \vdash \hat{\tau}_1 \Leftarrow \texttt{Ty} \rightsquigarrow \tau_1 \dashv \Delta_1 \qquad \Phi \vdash \hat{\tau}_2 \Leftarrow \texttt{Ty} \rightsquigarrow \tau_2 \dashv \Delta_2}{\Phi \vdash \hat{\tau}_1 \oplus \hat{\tau}_2 \Rightarrow \texttt{S}(\tau_1 \oplus \tau_2) \rightsquigarrow \tau_1 \oplus \tau_2 \dashv \Delta_1 \cup \Delta_2}$$

$$\texttt{TElabSList}\quad \frac{\Phi \vdash \hat{\tau} \Leftarrow \texttt{Ty} \rightsquigarrow \tau \dashv \Delta}{\Phi \vdash \texttt{list}(\hat{\tau}) \Rightarrow \texttt{S}(\texttt{list}(\tau)) \rightsquigarrow \texttt{list}(\tau) \dashv \Delta}$$

$$\texttt{TElabSVar}\quad \frac{t : \kappa \in \Phi}{\Phi \vdash t \Rightarrow \texttt{S}(t) \rightsquigarrow t \dashv \cdot}$$

$$\texttt{TElabSUVar}\quad \frac{t \notin \mathsf{dom}(\Phi)}{\Phi \vdash t \Rightarrow \texttt{KHole} \rightsquigarrow (\!|t|\!)^u \dashv u :: \texttt{KHole}}$$

$$\texttt{TElabSHole}\quad \frac{}{\Phi \vdash (\!|\,|\!)^u \Rightarrow \texttt{KHole} \rightsquigarrow (\!|\,|\!)^u \dashv u :: \texttt{KHole}}$$

$$\texttt{TElabSNEHole}\quad \frac{\Phi \vdash \hat{\tau} \Rightarrow \kappa \rightsquigarrow \tau \dashv \Delta}{\Phi \vdash (\!|\hat{\tau}|\!)^u \Rightarrow \texttt{KHole} \rightsquigarrow (\!|\tau|\!)^u \dashv \Delta, u :: \texttt{KHole}}$$

$\boxed{\Phi \vdash \hat{\tau} \Leftarrow \kappa \rightsquigarrow \tau \dashv \Delta}$ $\hat{\tau}$ analyzes against kind κ_1 and elaborates to τ

$$\texttt{TElabASubsume}\quad \frac{\hat{\tau} \neq (\!|\,|\!)^u \qquad \hat{\tau} \neq (\!|\hat{\tau}'|\!)^u \qquad \Phi \vdash \hat{\tau} \Rightarrow \kappa' \rightsquigarrow \tau \dashv \Delta \qquad \Delta; \Phi \vdash \kappa' \lesssim \kappa}{\Phi \vdash \hat{\tau} \Leftarrow \kappa \rightsquigarrow \tau \dashv \Delta}$$

$$\texttt{TElabAEHole}\quad \frac{}{\Phi \vdash (\!|\,|\!)^u \Leftarrow \kappa \rightsquigarrow (\!|\,|\!)^u \dashv u :: \kappa}$$

$$\texttt{TElabANEHole}\quad \frac{\Phi \vdash \hat{\tau} \Rightarrow \kappa' \rightsquigarrow \tau \dashv \Delta}{\Phi \vdash (\!|\hat{\tau}|\!)^u \Leftarrow \kappa \rightsquigarrow (\!|\tau|\!)^u \dashv \Delta, u :: \kappa}$$

$\boxed{\Phi_1 \vdash \tau : \kappa \rhd \rho \dashv \Phi_2}$ ρ matches against $\tau : \kappa$ extending Φ if necessary

$$\texttt{RESVar}\quad \frac{t \ \texttt{valid}}{\Phi \vdash \tau : \kappa \rhd t \dashv \Phi, t :: \kappa}$$

$$\texttt{RESEHole}\quad \frac{}{\Phi \vdash \tau : \kappa \rhd (\!|\,|\!) \dashv \Phi}$$

$$\texttt{RESVarHole}\quad \frac{\neg(t \ \texttt{valid})}{\Phi \vdash \tau : \kappa \rhd (\!|t|\!) \dashv \Phi}$$

$\boxed{\Gamma; \Phi \vdash e \Rightarrow \hat{\tau} \rightsquigarrow d \dashv \Delta}$ e synthesizes type τ and elaborates to d

$$\texttt{ESDefine}\ \frac{\Phi_1 \vdash \hat{\tau} \Rightarrow \kappa \rightsquigarrow \tau \dashv \Delta_1 \qquad \Phi_1 \vdash \tau : \kappa \rhd \rho \dashv \Phi_2 \qquad \Gamma; \Phi_2 \vdash e \Rightarrow \tau_1 \rightsquigarrow d \dashv \Delta_2}{\Gamma; \Phi_1 \vdash \texttt{type}\ \rho\ \texttt{=}\ \hat{\tau}\ \texttt{in}\ e \Rightarrow \tau_1 \rightsquigarrow \texttt{type}\ \rho\ \texttt{=}\ \tau : \kappa\ \texttt{in}\ d \dashv \Delta_1 \cup \Delta_2}$$

$\boxed{\Delta; \Gamma; \Phi \vdash d : \tau}$ d is assigned type τ

$$\texttt{DEDefine}\ \frac{\Phi_1 \vdash \tau_1 : \kappa \rhd \rho \dashv \Phi_2 \qquad \Delta; \Gamma; \Phi_2 \vdash d : \tau_2}{\Delta; \Gamma; \Phi_1 \vdash \texttt{type}\ \rho\ \texttt{=}\ \tau_1 : \kappa\ \texttt{in}\ d : \tau_2}$$

Theorem 1 (Well-Kinded Elaboration)
(1) If $\Phi \vdash \hat{\tau} \Rightarrow \kappa \rightsquigarrow \tau \dashv \Delta$ then $\Delta; \Phi \vdash \tau \Rightarrow \kappa$
(2) If $\Phi \vdash \hat{\tau} \Leftarrow \kappa \rightsquigarrow \tau \dashv \Delta$ then $\Delta; \Phi \vdash \tau \Leftarrow \kappa$

This is like the Typed Elaboration theorem in the POPL19 paper.

Theorem 2 (Elaborability)
(1) $\exists\Delta$ s.t. if $\Delta; \Phi \vdash \tau \Rightarrow \kappa$ then $\exists\hat{\tau}$ such that $\Phi \vdash \hat{\tau} \Rightarrow \kappa \rightsquigarrow \tau \dashv \Delta$
(2) $\exists\Delta$ s.t. if $\Delta; \Phi \vdash \tau \Leftarrow \kappa$ then $\exists\hat{\tau}$ such that $\Phi \vdash \hat{\tau} \Leftarrow \kappa \rightsquigarrow \tau \dashv \Delta$

This is similar but a little different from Elaborability theorem in the POPL19 paper. Choose the Δ that is emitted from elaboration and then there's an $\hat{\tau}$ that elaborates to any of the τ forms. Elaborability and Well-Kinded Elaboration implies we can just rely on the elaboration forms for the premises of any rules that demand kind synthesis/analysis.

Theorem 3 (Type Elaboration Unicity)
(1) If $\Phi \vdash \hat{\tau} \Rightarrow \kappa_1 \rightsquigarrow \tau_1 \dashv \Delta_1$ and $\Phi \vdash \hat{\tau} \Rightarrow \kappa_2 \rightsquigarrow \tau_2 \dashv \Delta_2$ then $\kappa_1 = \kappa_2$, $\tau_1 = \tau_2$, $\Delta_1 = \Delta_2$
(2) If $\Phi \vdash \hat{\tau} \Leftarrow \kappa \rightsquigarrow \tau_1 \dashv \Delta_1$ and $\Phi \vdash \hat{\tau} \Leftarrow \kappa \rightsquigarrow \tau_2 \dashv \Delta_2$ then $\tau_1 = \tau_2$, $\Delta_1 = \Delta_2$

This is like the Elaboration Unicity theorem in the POPL19 paper.

Attachment 1

Theorem 4 (Kind Synthesis Precision)
If $\Delta; \Phi \vdash \tau \Rightarrow \kappa_1$ *and* $\Delta; \Phi \vdash \tau \Leftarrow \kappa_2$ *then* $\Delta; \Phi \vdash \kappa_1 \lesssim \kappa_2$

Kind Synthesis Precision says that synthesis finds the most precise kappa possible for a given input type. This is somewhat trivial, but interesting to note because it means we can expect singletons wherever possible.

Theorem 5.1 (Kind Analysis Soundness)

If $\Delta; \Phi \vdash \tau \Leftarrow \kappa$ then $\Delta; \Phi \vdash \tau :: \kappa$

Theorem 5.2 (Kind Analysis Completeness)

If $\Delta; \Phi \vdash \tau :: \kappa$ then $\Delta; \Phi \vdash \tau \Leftarrow \kappa$

5.1 Induction on complexity

Base cases

$c :: Ty$ & $\kappa = Ty \Rightarrow Ty \leq Ty \Rightarrow c :: \kappa$

or

$c \Leftarrow \kappa \Rightarrow c \Rightarrow \kappa'$ & $\kappa' \lesssim \kappa \Rightarrow \kappa' = S(c) \Rightarrow S(c) \lesssim \kappa \Rightarrow S(c) \equiv \kappa \Rightarrow c :: S(c)$ & $S(c) \leq \kappa \Rightarrow c :: \kappa$

$t \Leftarrow \kappa \Rightarrow t \Rightarrow \kappa'$ & $\kappa' \lesssim \kappa \Rightarrow \kappa' = S_{\kappa''}(t)$ & $t :: \kappa'' \in \Phi \Rightarrow t :: \kappa'' \Rightarrow \kappa = Ty \Rightarrow t :: Ty \Rightarrow t :: \kappa$ $\quad t :: S_{\kappa''}(t) \quad S_{\kappa''}(t) \lesssim \kappa$

or

$\kappa \equiv S_{\kappa''}(t) \Rightarrow S_{\kappa''}(t) \lesssim \kappa \Rightarrow t :: S_{\kappa''}(t)$ $\quad t :: \kappa$

$\underset{\sim}{t} \Leftarrow \kappa$

$(\!)^u \Leftarrow \kappa$