

Hazel MetaPhi: 9-type-aliases

July 3, 2021

introduction

f

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 premised on $\lesssim (\Delta; \Phi \vdash \kappa_1 \lesssim \kappa_2)$ at **KAASubsume**, which is itself premised on $\equiv (\Delta; \Phi \vdash \tau_1 \equiv \tau_2)$ at **KCRespectEquiv**, which is itself premised on $:: (\Delta; \Phi \vdash \tau :: \kappa)$ at **KCESingEquiv**.

Explicitly algorithmic counterparts to \lesssim and \equiv need to be defined. Suggested notation: \trianglelefteq and $\trianglelefteq\Rightarrow$ (faintly reminiscent of Stone and Harper).

Issue 2: A $t :: \kappa \in \Phi \implies t :: \mathbf{S}(t)$ (like) rule is problematic

When $\forall \tau. \Delta; \Phi \vdash \tau :: \kappa \implies \Delta; \Phi \vdash \tau :: \mathbf{Type}$, as is now, this is fine (except for improperly (non)contexted holes $\forall \tau. \vdash \tau :: \mathbf{Type}$ 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 **list**(τ) to be a type constructor proper instead of a builtin type schema, such that $\vdash \mathbf{list} :: \mathbf{Type} \rightarrow \mathbf{Type}$, and had nonstifled type aliases such that $t :: \mathbf{list}$, $\mathbf{S}(t)$ 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 language 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 metalanguage. **HOSingletons** should make the metatheory proofs more general. 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):

$$\vdash \lambda t::\text{Type}.t \stackrel{\text{S}(\text{Int}) \rightarrow \text{Type}}{\equiv} \lambda t::\text{Type}.\text{Int}$$

should be derivable but

$$\vdash \lambda t::\text{Type}.t \stackrel{\text{Type} \rightarrow \text{Type}}{\equiv} \lambda t::\text{Type}.\text{Int}$$

should not, where $\vdash \text{Type} \rightarrow \text{Type} \lesssim \text{S}(\text{Int}) \rightarrow \text{Type}$
 Interestingly

$$\vdash \lambda t::\text{Type}.t \stackrel{\text{S}(\text{Int}) \rightarrow \text{S}(\text{Int})}{\equiv} \lambda t::\text{Type}.\text{Int}$$

should also be derivable, where $\vdash \text{S}(\text{Int}) \rightarrow \text{S}(\text{Int}) \lesssim \text{S}(\text{Int}) \rightarrow \text{Type}$
 With “real” subkinding, this behavior is more serious.

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

$$\vdash \lambda x:\text{Int}.\text{if } x \geq 0 \text{ then } x \text{ else } 2 * x \stackrel{\text{Nat} \rightarrow \text{Int}}{\equiv} \lambda x:\text{Int}.x$$

should be derivable but

$$\vdash \lambda x:\text{Int}.\text{if } x \geq 0 \text{ then } x \text{ else } 2 * x \stackrel{\text{Int} \rightarrow \text{Int}}{\equiv} \lambda x:\text{Int}.x$$

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

$\text{BinOp } \oplus ::= \text{Product} \mid \text{Sum} \mid \text{Arrow}$
 $\text{Kind } \kappa ::= \text{Ty} \mid \text{KHole} \mid \text{S}(\tau)$
 $\text{ConstantTypes } c ::= \text{Int} \mid \text{Float} \mid \text{Bool}$
 $\text{UserHTyp } \hat{\tau} ::= c \mid \hat{\tau}_1 \oplus \hat{\tau}_2 \mid \text{list}(\hat{\tau}) \mid \langle \rangle^u \mid \langle \hat{\tau} \rangle^u$
 $\text{InternalHTyp } \tau ::= c \mid \tau_1 \oplus \tau_2 \mid \text{list}(\tau) \mid \langle \rangle^u \mid \langle \tau \rangle^u$ *10 tD*
 $\text{TypeVars } t$
 $\text{TypePattern } \rho ::= t \mid \langle \rangle \mid \langle t \rangle$
 $\text{UserExpression } e ::= \text{type } \rho = \hat{\tau} \text{ in } e \mid \text{elided}$
 $\text{InternalExpression } \tau ::= \text{type } \rho = \tau : \kappa \text{ in } d \mid \text{elided}$

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

KCHoleL

$\frac{}{\Delta; \Phi \vdash \text{KHole} \lesssim \kappa}$

KCHoleR

$\frac{}{\Delta; \Phi \vdash \kappa \lesssim \text{KHole}}$

KCRespectEquiv

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

KCSubsumption

$\frac{\Delta; \Phi \vdash \tau \not\Leftarrow \text{Ty}}{\Delta; \Phi \vdash \text{S}(\tau) \lesssim \text{Ty}}$ *7*

$\boxed{t \text{ valid}}$ 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.

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

KFTy

$\frac{}{\Delta; \Phi \vdash \text{Ty} \text{ kind}}$

KFHole

$\frac{}{\Delta; \Phi \vdash \text{KHole} \text{ kind}}$

KFSing

$\frac{\Delta; \Phi \vdash \tau \not\Leftarrow \text{Ty}}{\Delta; \Phi \vdash \text{S}(\tau) \text{ kind}}$ *43.2a*

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

$$\begin{array}{c} \text{KESymm} \\ \Delta; \Phi \vdash \kappa_1 \equiv \kappa_2 \\ \hline \Delta; \Phi \vdash \kappa_2 \equiv \kappa_1 \end{array} \quad \begin{array}{c} \text{KETrans} \\ \Delta; \Phi \vdash \kappa_1 \equiv \kappa_2 \quad \Delta; \Phi \vdash \kappa_2 \equiv \kappa_3 \\ \hline \Delta; \Phi \vdash \kappa_1 \equiv \kappa_3 \end{array}$$

$$\begin{array}{c} \text{KESingEquiv} \\ \Delta; \Phi \vdash \tau_1 \equiv \tau_2 \\ \hline \Delta; \Phi \vdash S(\tau_1) \equiv S(\tau_2) \end{array}$$

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

*would need to define labelled singletors for KHole
p.687 St. Harp 2006*

$$\begin{array}{c} \text{KSConst} \\ \Delta; \Phi \vdash c \Rightarrow S(c) \end{array} \quad \begin{array}{c} \text{KSVar} \\ t : \kappa \in \Phi \\ \hline \Delta; \Phi \vdash t \Rightarrow S(t) \end{array} \quad \begin{array}{c} \text{KSUVar} \\ t \notin \text{dom}(\Phi) \quad u :: \kappa \in \Delta \\ \hline \Delta; \Phi \vdash t \Rightarrow \text{KHole} \end{array}$$

$$\begin{array}{c} \text{KSBinOp} \\ \Delta; \Phi \vdash \tau_1 \Leftarrow S(\tau_1) \quad \Delta; \Phi \vdash \tau_2 \Leftarrow S(\tau_2) \\ \hline \Delta; \Phi \vdash \tau_1 \oplus \tau_2 \Rightarrow S(\tau_1 \oplus \tau_2) \end{array}$$

$$\begin{array}{c} \text{KSList} \\ \Delta; \Phi \vdash \tau \Leftarrow S(\tau) \\ \hline \Delta; \Phi \vdash \text{list}(\tau) \Rightarrow S(\text{list}(\tau)) \end{array} \quad \begin{array}{c} \text{KSEHole} \\ u :: \kappa \in \Delta \\ \hline \Delta; \Phi \vdash ()^u \Rightarrow \kappa \end{array}$$

$$\begin{array}{c} \text{KSNEHole} \\ u :: \kappa \in \Delta \quad \Delta; \Phi \vdash \tau \Rightarrow \kappa' \\ \hline \Delta; \Phi \vdash ()^u \Rightarrow \kappa \end{array}$$

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

$$\begin{array}{c} \text{KAASubsume} \\ \Phi \vdash \tau \Rightarrow \kappa' \quad \Delta; \Phi \vdash \kappa' \lesssim \kappa \\ \hline \Delta; \Phi \vdash \tau \Leftarrow \kappa \end{array}$$

KAVar
 $t :: \kappa_1 \in \Phi \quad \kappa_1 \lesssim \kappa$
 $\Delta; \Phi \vdash t \Leftarrow \kappa$

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

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

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

$$\frac{\text{KCESingEquiv} \quad \Delta; \Phi \vdash \tau_1 \not\equiv S(\tau_2)}{\Delta; \Phi \vdash \tau_1 \equiv \tau_2}$$

$$\frac{\text{KCERefl} \quad \tau :: k}{\Delta; \Phi \vdash \tau \equiv \tau}$$

$$\frac{\text{KCEConst} \quad \Delta; \Phi \vdash c \equiv c}{\Delta; \Phi \vdash c \equiv c}$$

$$\frac{\text{KCEVar} \quad t : \kappa \in \Phi}{\Delta; \Phi \vdash t \equiv t}$$

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

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

$$\frac{\text{KCEEHole} \quad u :: \kappa \in \Delta}{\Delta; \Phi \vdash \langle \rangle^u \equiv \langle \rangle^u}$$

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

$$\frac{\text{KCENEHole} \quad u :: \kappa \in \Delta \quad \Delta; \Phi \vdash \tau \not\equiv \kappa'}{\Delta; \Phi \vdash \langle \tau \rangle^u \equiv \langle \tau \rangle^u}$$

* sim

$$\Delta; \Phi \vdash \tau :: k$$

τ is well formed at kind k

$$\frac{t :: k \in \Xi}{\Delta; \Phi \vdash t :: k} \text{ 17}$$

$$\frac{\Delta; \Phi \vdash \tau :: \tau_g}{\Delta; \Phi \vdash \text{list}(\tau) :: \tau_g}$$

$$\frac{\Delta; \Phi \vdash \tau :: k_1 \quad \Delta; \Phi \vdash k_1 \leq k}{\Delta; \Phi \vdash \tau :: k} \text{ 43.2d}$$

$$\frac{\Delta; \Phi \vdash \tau :: \tau_g \quad 23}{\Delta; \Phi \vdash \tau :: S(\tau)} \text{ 43.2b}$$

$$\frac{u :: k \in \Delta}{\Delta; \Phi \vdash \langle \rangle^u :: k}$$

$$\frac{\Delta; \Phi \vdash u :: k \in \Delta \quad \Delta; \Phi \vdash \tau :: k_1}{\Delta; \Phi \vdash \langle \tau \rangle^u :: k}$$

$$\frac{\Delta; \Phi \vdash \tau :: k}{\Delta; \Phi \vdash \tau :: S_k(\tau)} \text{ replaces 23?}$$

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

$$\frac{\Delta; \Phi \vdash \tau_1 :: \tau_g \quad \Delta; \Phi \vdash \tau_2 :: \tau_g}{\Delta; \Phi \vdash \tau_1 \oplus \tau_2 :: \tau_g}$$

$$\frac{t \notin \text{dom}(\Xi) \quad u :: k \in \Delta}{\Delta; \Phi \vdash \langle t \rangle^u :: k}$$

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

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

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

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

TElabSConst

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

TElabSBinOp

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

TElabSList

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

TElabSVar

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

TElabSUNVar

$$\frac{t \notin \text{dom}(\Phi)}{\Phi \vdash t \Rightarrow \text{KHole} \rightsquigarrow (\textcolor{violet}{t})^u \dashv u :: \text{KHole}}$$

TElabSHole

$$\frac{}{\Phi \vdash (\textcolor{violet}{\text{Hole}})^u \Rightarrow \text{KHole} \rightsquigarrow (\textcolor{violet}{\text{Hole}})^u \dashv u :: \text{KHole}}$$

TElabSNEHole

$$\frac{\Phi \vdash \hat{\tau} \Rightarrow \kappa \rightsquigarrow \tau \dashv \Delta}{\Phi \vdash (\hat{\tau})^u \Rightarrow \text{KHole} \rightsquigarrow (\textcolor{violet}{\tau})^u \dashv \Delta, u :: \text{KHole}}$$

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

TElabASubsume

$$\frac{\hat{\tau} \neq (\textcolor{violet}{\text{Hole}})^u \quad \hat{\tau} \neq (\hat{\tau}')^u \quad \Phi \vdash \hat{\tau} \Rightarrow \kappa' \rightsquigarrow \tau \dashv \Delta \quad \Delta; \Phi \vdash \kappa' \lesssim \kappa}{\Phi \vdash \hat{\tau} \Leftarrow \kappa \rightsquigarrow \tau \dashv \Delta}$$

TElabAEHole

$$\frac{}{\Phi \vdash (\textcolor{violet}{\text{Hole}})^u \Leftarrow \kappa \rightsquigarrow (\textcolor{violet}{\text{Hole}})^u \dashv u :: \kappa}$$

TElabANEHole

$$\frac{\Phi \vdash \hat{\tau} \Rightarrow \kappa' \rightsquigarrow \tau \dashv \Delta}{\Phi \vdash (\hat{\tau})^u \Leftarrow \kappa \rightsquigarrow (\textcolor{violet}{\tau})^u \dashv \Delta, u :: \kappa}$$

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

RESVar

$$\frac{t \text{ valid}}{\Phi \vdash \tau : \kappa \triangleright t \dashv \Phi, t :: \kappa}$$

RESEHole

$$\frac{}{\Phi \vdash \tau : \kappa \triangleright (\textcolor{blue}{\text{Hole}}) \dashv \Phi}$$

RESVarHole

$$\frac{\neg(t \text{ valid})}{\Phi \vdash \tau : \kappa \triangleright (\textcolor{blue}{t}) \dashv \Phi}$$

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

ESDefine

$$\frac{\begin{array}{c} \Phi_1 \vdash \hat{\tau} \Rightarrow \kappa \rightsquigarrow \tau \dashv \Delta_1 \\ \Phi_1 \vdash \tau : \kappa \triangleright \rho \dashv \Phi_2 \quad \Gamma; \Phi_2 \vdash e \Rightarrow \tau_1 \rightsquigarrow d \dashv \Delta_2 \end{array}}{\Gamma; \Phi_1 \vdash \mathbf{type} \ \rho = \hat{\tau} \ \mathbf{in} \ e \Rightarrow \tau_1 \rightsquigarrow \mathbf{type} \ \rho = \tau : \kappa \ \mathbf{in} \ d \dashv \Delta_1 \cup \Delta_2}$$

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

DEDefine

$$\frac{\begin{array}{c} \Phi_1 \vdash \tau_1 : \kappa \triangleright \rho \dashv \Phi_2 \quad \Delta; \Gamma; \Phi_2 \vdash d : \tau_2 \end{array}}{\Delta; \Gamma; \Phi_1 \vdash \mathbf{type} \ \rho = \tau_1 : \kappa \ \mathbf{in} \ d : \tau_2}$$

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.

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 $\Delta; \Phi \vdash \tau \Leftarrow \kappa$

S.1 Induction on complexity

Base cases

$$C :: T_g \text{ i } K \Rightarrow T_g \leq T_g \Rightarrow C :: K$$

$$C \Leftarrow K \Rightarrow C \exists K' \text{ i } K' \leq K \Rightarrow K' = S(C) \Rightarrow S(C) \leq K \Rightarrow S(C) \equiv K \Rightarrow C :: S(C) \text{ i } S(C) \leq K \Rightarrow C :: K$$

$$t \Leftarrow K \Rightarrow t \exists K' \text{ i } K' \leq K \Rightarrow K' = S_{K'}(t) \text{ i } t :: K' \leq K \Rightarrow t :: K'' \Rightarrow K = T_g \Rightarrow t :: T_g \Rightarrow t :: K$$

$$\underline{t} \Leftarrow K$$

$$\text{OD}'' \Leftarrow K$$

$$K \equiv S_{K'}(t) \Rightarrow S_{K''}(t) \leq K \Rightarrow t :: S_{K'}(t) \quad t :: K$$