Inhabitation of low-rank interstcion types with subtyping. NOTES

Ilya Kaysin

February, 2019

1 Introduction

1.1 System $\lambda_{\wedge\eta}$

Inference rules: Ax, $I \to$, $E \to$, $I\wedge$, $E\wedge$, η.

$$\frac{}{\Gamma, x\colon \tau \vdash x\colon \tau}\ (Ax)$$

$$\frac{\Gamma, x\colon \sigma \vdash M\colon \tau}{\Gamma \vdash (\lambda x\colon \sigma M)\colon \sigma \to \tau}\ (I \to)$$

$$\frac{\Gamma \vdash M\colon \sigma \to \tau \quad \Gamma \vdash N\colon \sigma}{\Gamma \vdash (MN)\colon \tau}\ (E \to)$$

$$\frac{\Gamma \vdash M\colon \sigma \quad \Gamma \vdash M\colon \tau}{\Gamma \vdash M\colon \sigma \wedge \tau}\ (I\wedge)$$

$$\frac{\Gamma \vdash M\colon \sigma \wedge \tau}{\Gamma \vdash M\colon \sigma}\ (E\wedge)$$

$$\frac{\Gamma \vdash M\colon \sigma}{\Gamma \vdash (\lambda x\colon Mx)\colon \sigma}\ (\eta)$$

TODO: add definitions

In some works that system is called $\lambda_\wedge$, but we denote it by $\lambda_{\wedge\eta}$ to distinguish with the system without η-rule which we denote by $\lambda_\wedge$.

Thus, we have two operations to form types: $\to$ and $\wedge$.

1.2 Subtyping

Rules $E\wedge$ and η could be replaced by the $\leqslant$ rule:

$$\frac{\Gamma \vdash M : \sigma \quad \sigma \leqslant \tau}{\Gamma \vdash M : \tau}$$

where $\leqslant$ is defined as follows:

$$\sigma \leqslant \sigma$$

$$\sigma \leqslant \sigma \wedge \sigma$$

$$\sigma \wedge \tau \leqslant \sigma$$

$$\sigma \wedge \tau \leqslant \tau$$

$$(\sigma \to \tau_1) \wedge (\sigma \to \tau_2) \leqslant \sigma \to (\tau_1 \wedge \tau_2)$$

$$\frac{\sigma \leqslant \sigma' \quad \tau \leqslant \tau'}{\sigma \wedge \tau \leqslant \sigma' \wedge \tau'}$$

$$\frac{\sigma \leqslant \sigma' \quad \tau \leqslant \tau'}{\sigma' \to \tau \leqslant \sigma \to \tau'}$$

$$\frac{\tau_1 \leqslant \tau_2 \quad \tau_2 \leqslant \tau_3}{\tau_1 \leqslant \tau_3}$$

It is shown in [Hindley 82] that relation $\leqslant$ is recursive and it's possible for any two types α and β to determine if $\alpha \leqslant \beta$ by the following algorithm:

```haskell
(<:) :: Type -> Type -> Bool
TVar a       <: TVar b       = a == b
TVar _       <: (_ :-> _)  = False
(_ :-> _) <: TVar _       = False
(sA :-> tA) <: (sB :-> tB) = (tA <: tB) && (sB <: sA)
a            <: b            = all (\t -> any (<:t) a') b' where
                                 a' = removeIntersection a
                                 b' = removeIntersection b
                                 removeIntersection atom@(TVar _) = [atom]
                                 removeIntersection arrow@(_ :-> _) = [arrow]
                                 removeIntersection (a :^: b)        = [a, b] >>= removeIntersection
```

TODO: translate to more clear pseudocode

Note that we use $\wedge$ only as a tool during the typing process, but not within types themselves. Since the systems $\lambda_{\wedge\eta}$ and $\lambda_{\wedge\leqslant}$ are equivalent, we'll denote them both as $\lambda_{\wedge\eta}$.

Definition 1. $\sigma \sim \tau \iff \sigma \leqslant \tau$ and $\tau \leqslant \sigma$

Lemma 1. $\alpha \to \beta \wedge \gamma \sim (\alpha \to \beta) \wedge (\alpha \to \gamma)$

Proof. We have to prove two statements. $(\alpha \to \beta) \wedge (\alpha \to \gamma) \leqslant \alpha \to \beta \wedge \gamma$ is just an axiom.

$$\dfrac{\alpha \to \beta \wedge \gamma \leqslant \alpha \to \beta \wedge \gamma \qquad \dfrac{\dfrac{\alpha \leqslant \alpha \quad \beta \wedge \gamma \leqslant \beta}{\alpha \to \beta \wedge \gamma \leqslant \alpha \to \beta} \quad \dfrac{\alpha \leqslant \alpha \quad \beta \wedge \gamma \leqslant \gamma}{\alpha \to \beta \wedge \gamma \leqslant \alpha \to \gamma}}{(\alpha \to \beta \wedge \gamma) \wedge (\alpha \to \beta \wedge \gamma) \leqslant (\alpha \to \beta) \wedge (\alpha \to \gamma)}}{\alpha \to \beta \wedge \gamma \leqslant (\alpha \to \beta) \wedge (\alpha \to \gamma)}$$

□

Definition 2. Let A and B be sets of types. Then we override $\to$ and $\wedge$ as follows:

$$A \to B = \{\alpha \to \beta \mid \alpha \in A, \beta \in B\}$$

$$A \wedge B = \{\alpha \wedge \beta \mid \alpha \in A, \beta \in B\}$$

In addition, if one of the arguments of $\to$ or $\wedge$ is a type τ we consider this argument as one-element set $\{\tau\}$.

Definition 3. Denote a set of all subtypes of type τ by $\underline{\tau}$, and a set of all supertypes of τ by $\bar{\tau}$

Lemma 2. $\underline{\sigma \to \tau} = \varphi \wedge (\bar{\sigma} \to \underline{\tau})$, *(for some (possibly empty) φ)*

Lemma 3. *Let σ and τ be types. The following three statements are equivalent:*

1. $\sigma \leqslant \tau$

2. $x \colon \sigma \vdash x \colon \tau$

3. *For every environment Γ and term t such that $\Gamma \vdash t \colon \sigma$ there is an inference tree:* $\begin{array}{c}\Gamma \vdash t \colon \sigma \\ \vdots \\ \Gamma \vdash t \colon \tau\end{array}$

1.3 $\lambda_{\wedge\eta}$ VS $\lambda_{\wedge}$

Does the $\leqslant$-rule make a big deal? Let's consider an example. It could be proved that typing statement $x : \alpha \to \beta \wedge \gamma \vdash x : \alpha \to \beta$ is true in $\lambda_{\wedge\eta}$ but false in $\lambda_{\wedge}$.

It entails that the type $\delta \wedge (\alpha \to \beta \wedge \gamma) \to \delta \wedge (\alpha \to \beta)$ is empty in $\lambda_{\wedge}$ and contains term `id` in $\lambda_{\wedge\eta}$.

Note that the type $(\alpha \to \beta \wedge \gamma) \to \alpha \to \beta$ isn't empty in both systems, but in $\lambda_{\wedge}$ it contains only $\lambda ab.ab$ (doesn't contain $\lambda a.a$).

2 Inhabitation algorithm

The inhabitation algorithm is a modification of an algorithm described in [Kuśmierek 2007] for $\lambda_{\wedge}$ system. Frankly speaking, the algorithm and the proof of it's correctness contain a mistake, and we'll fix it. Although, ironically, with this mistake the algorithm turns into an algorithm for $\lambda_{\wedge\eta}$.

2.1 Algorithm for $\lambda_\wedge$

At each moment we are looking for a single solution X of a system of constrains: $\Gamma_1 \vdash X : \tau_1, \dots, \Gamma_n \vdash X : \tau_n$, the domains of all Γ_i are the same. Then we perform the following transformations with the system until it's empty.

- 1. One of the types τ_i is an intersection ($\tau_i = \sigma \wedge \rho$). Then the constrain $\Gamma_i \vdash X : \tau_i$ is replaced by two: $\Gamma_i \vdash X : \sigma$ and $\Gamma_i \vdash X : \rho$.
- 2. Every τ_i is an arrow ($\tau_i = \sigma_i \to \rho_i$). Then we say that $X = \lambda x.X'$ where X' is a solution of the system $\Gamma_1, x : \sigma_1 \vdash X' : \rho_1, \dots, \Gamma_n, x : \sigma_n \vdash X' : \rho_n$
- $3_\wedge$. One of the types τ_i is a type variable. Then X can't be an abstraction and has to be an application. So we pick a variable x and a number $k \geq 0$ such that $\Gamma_i \vdash \lambda z_1 \dots z_k.xz_1 \dots z_k : \rho_i^1 \to \dots \to \rho_i^k \to \tau_i$, for each i. Then we say that $X = xZ^1 \dots Z^k$ where Z^k is a solution of the system $\Gamma_1 \vdash Z^k : \rho_1^k, \dots, \Gamma_n \vdash Z^k : \rho_n^k$. K systems could be independently solved in parallel.

2.2 Algorithm for $\lambda_{\wedge\eta}$

The algorithm is basically the same as the previous one. The only difference is in the transformation 3:

- $3_{\wedge\eta}$. One of the types τ_i is a type variable. Then X can't be an abstraction and has to be an application. So we pick a variable x and a number $k \geq 0$ such that $\Gamma_i \vdash x : \rho_i^1 \to \dots \to \rho_i^k \to \tau_i$, for each i. Then we say that $X = xZ^1 \dots Z^k$ where Z^k is a solution of the system $\Gamma_1 \vdash Z^k : \rho_1^k, \dots, \Gamma_n \vdash Z^k : \rho_n^k$. K systems could be independently solved in parallel.

2.3 $\lambda_{\wedge\eta}$ VS $\lambda_\wedge$ in terms of algorithm

The algorithm for $\lambda_\wedge$ in [Kuśmierek 2007] is mistakenly the same as our algorithm for $\lambda_{\wedge\eta}$ above. Let's consider an example where it doesn't work in $\lambda_\wedge$.

The task is $p\colon \alpha \to \beta \wedge \gamma, q\colon \alpha \vdash X\colon \beta$. The algorithm goes into the transformation 3. But there's no such x in the context that $p\colon \alpha \to \beta \wedge \gamma, q\colon \alpha \vdash x\colon \dots \to \beta$, because, as we mentioned in the Introduction, $p\colon \alpha \to \beta \wedge \gamma \vdash p\colon \alpha \to \beta$ is false in $\lambda_\wedge$. But $p\colon \alpha \to \beta \wedge \gamma, q\colon \alpha \vdash \lambda z.pz\colon \alpha \to \beta$, because, intuitively, the λ allows us to apply $(I \to)$ and remove $\to$,

2.4 Soundness and completeness

Following definitions and statements are our interpretation and correction of the corresponding section in [Kuśmierek 2007].

Lemma 4. *If the algorithm finds a term M for an input type τ, then $\vdash M\colon \tau$ in the corresponding type system.*

Lemma 5. *(In system $\lambda_\wedge$)*
Let x be a variable, M_i be a term, τ be a type without intersections on the top level.
If $\Gamma \vdash xM_1 \dots M_k\colon \tau$ then $\Gamma \vdash \lambda z_1 \dots z_k.xz_1 \dots z_k\colon \alpha_1 \to \dots \to \alpha_k \to \tau$

Proof. If $k = 0$ then the statement is obvious.

Let's look at the inference rules and try to rebuild the inference tree from the end to the beginning. Which rule could we use the last? Obviously, we couldn't use $(I \to)$, since the term isn't a λ, we couldn't use $(I\wedge)$, since τ isn't an intersection, we couldn't use the axiom Ax, since the term isn't a variable. So the picture is the following: we applied a sequence of $(E\wedge)$ and $(I\wedge)$, then in one of the branches we got $\Gamma \vdash xM_1 \dots M_k\colon \sigma \wedge \tau$, and then we applied $(E \to)$ to it:

$$\cfrac{\Gamma \vdash xM_1 \dots M_{k-1}\colon \alpha_k \to \varphi_k \wedge \tau \quad \Gamma \vdash M_k\colon \alpha_k}{\Gamma \vdash xM_1 \dots M_k\colon \varphi_k \wedge \tau}$$
$$\vdots$$
$$\Gamma \vdash xM_1 \dots M_k\colon \tau$$

(for some (possibly empty) φ_i)

Note that the task $\Gamma \vdash xM_1 \dots M_{k-1} \colon \alpha_1 \to \varphi_1 \wedge \tau$ satisfies the lemma's conditions, so we can iterate the process $k-1$ more times and get:

$$\Gamma \vdash x \colon \alpha_1 \to \varphi_1 \wedge (\alpha_2 \to \varphi_2 \wedge \dots (\alpha_k \to \varphi_k \wedge \tau) \dots)$$

Then we can perform the following procedure for $i = 1 \dots k$:

1. add fresh $z_i \colon \alpha_i$ to the context
2. apply $(E \to)$ (with $\Gamma, \dots, z_i \colon \alpha_i \vdash z_i \colon \alpha_i$)
3. apply $(E\wedge)$ to remove φ_i

Now $\Gamma, z_1 : \alpha_1 \dots z_k : \alpha_k \vdash xz_1 \dots z_k : \tau$ Applying $(I \to)$ k times, we get $\Gamma \vdash \lambda z_1 \dots z_k . xz_1 \dots z_k \colon \alpha_1 \to \dots \to \alpha_k \to \tau$

□

Note that since $\Gamma \vdash x \colon \alpha_1 \to \varphi_1 \wedge (\alpha_2 \to \varphi_2 \wedge \dots (\alpha_k \to \varphi_k \wedge \tau) \dots)$, further inference tree analysis reasoning leads us to the

Lemma 6. *(In system $\lambda_\wedge$)*
Let x be a variable, M_i be a term, τ be a type without intersections on the top level.
If $\Gamma \vdash xM_1 \dots M_k \colon \tau$ then
$\Gamma \ni x \colon \varphi_0 \wedge (\alpha_1 \to \varphi_1 \wedge (\alpha_2 \to \varphi_2 \wedge \dots (\alpha_k \to \varphi_k \wedge \tau)$ *(for some (possibly empty) φ_i)*

This lemma gives us an explicit algorithm for the transformation $3_\wedge$. We'll describe it later.

Lemma 7. *(In system $\lambda_{\wedge,\eta}$)*
Let x be a variable, M_i be a term, τ be a type without intersections on the top level. If $\Gamma \vdash xM_1 \dots M_k \colon \tau$ then $\Gamma \vdash x \colon \alpha_1 \to \dots \to \alpha_k \to \tau$.

Proof. The proof is almost the same as for $\lambda_\wedge$ — just inference tree analysis. Let's look at the lowest $(\to E)$. After that, unlike the tree, we can't use $(\wedge E)$ (because we don't have it in our system). We use $(\leqslant)$ instead.

$$\cfrac{\Gamma \vdash xM_1 \dots M_{k-1} \colon \alpha_k \to \underline{\tau} \quad \Gamma \vdash M_k \colon \alpha_k}{\begin{array}{c}\Gamma \vdash xM_1 \dots M_k \colon \underline{\tau} \\ \vdots \\ \Gamma \vdash xM_1 \dots M_k \colon \tau\end{array}}$$

(The notation is a bit informal: when we write $\Gamma \vdash a \colon A$ where A is a set of types we mean that the statement holds for some type from A)

Iterating the process $k-1$ more times we get:

$$\Gamma \vdash x \colon \alpha_1 \to \underline{(\alpha_2 \to \underline{\dots (\underline{\alpha_k \to \underline{\tau}}}}) \dots)$$

Since $\sigma \to \underline{\rho} \leqslant \sigma \to \rho$, induction gives us

$$\alpha_1 \to \underline{(\alpha_2 \to \underline{\dots (\underline{\alpha_k \to \underline{\tau}}}}) \dots) \leqslant \alpha_1 \to \dots \alpha_k \to \tau$$

Thus, applying $(\leqslant)$, $\Gamma \vdash x \colon \alpha_1 \to \dots \alpha_k \to \tau$ □

Note that since $\Gamma \vdash x \colon \alpha_1 \to \underline{(\alpha_2 \to \underline{\dots (\underline{\alpha_k \to \underline{\tau}}}}) \dots)$, further inference tree analysis reasoning leads us to the

Lemma 8. *(In system $\lambda_{\wedge,\eta}$)*
Let x be a variable, M_i be a term, τ be a type without intersections on the top level.
If $\Gamma \vdash xM_1 \dots M_k \colon \tau$ then
$\Gamma \ni x \colon \underline{\alpha_1 \to \underline{(\alpha_2 \to \underline{\dots (\underline{\alpha_k \to \underline{\tau}}}}}) \dots))$,
which is equivalent to
$\Gamma \ni x \colon \varphi_0 \wedge (\overline{\alpha_1} \to \varphi_1 \wedge (\overline{\alpha_2} \to \varphi_2 \wedge \dots (\overline{\alpha_k} \to \underline{\tau}) \dots))$ *(for some (possibly empty) φ_i)*

Proof. The equivalence might be non-obvious. We should apply lemma from the Section 1.2, keeping in mind that φ_i is any type, thus $\overline{\varphi_i}$ is also any type. □

This lemma gives us an explicit algorithm for the transformation $3_{\wedge\eta}$. We'll describe it later.

Definition 4. *M is a* long *solution of the task Z if the algorithm above finds it.*

Lemma 9. *If there exists a solution of a task Z, then there exists a long one.*

Proof. See Lemma 9 in [Kuśmierek 2007]. In this work they use lemmas above without proofs, but wrongly use the statement for $\lambda_{\wedge\eta}$ in $\lambda_{\wedge}$.

TODO: write proof, because in [Kuśmierek 2007] it might be a bit unclear. □

2.5 Explicit algorithm

The algorithms above have a problem: it's not really clear how to implement them. The transformations 1 and 2 are pretty obvious, but there are some questions about the transformation 3.

In both algorithms it's said that we non-deterministically choose k and x such that some type statement for x holds. But how to check algorithmically if that statement holds?

We propose an alternative method — to extract all possible x from the context for which the statement holds. To perform it let's define another opertation:

Definition 5.

$$unroll :: \texttt{Type} \to \texttt{Set of types}$$

$$unroll(x) = \{x\}\texttt{, if } x \texttt{ is a type variable}$$

$$unroll(\sigma \wedge \tau) = unroll(\sigma) \cup unroll(\tau)$$

$$unroll(\sigma \to \tau) = \{\sigma \to \rho \mid \rho \in unroll(\tau)\}$$

For example, $unroll(\alpha \to \gamma \wedge (\beta \to \gamma \wedge \delta)) = \{\alpha \to \beta \to \gamma, \alpha \to \beta \to \delta, \alpha \to \gamma\}$

Lemma 10. *For all $\psi \in unroll(\varphi) : \psi = \cdots \to \alpha$ where α is a type variable (or ψ might be a variable itself)*

Let's change the transformation $3_{\wedge\eta}$ to be explicit.

- $3^*_{\wedge\eta}$. One of the types to inhabit, suppose τ_j, is a type variable α. How to pick x and k such that $\Gamma_i \vdash x : \rho_i^1 \to \cdots \to \rho_i^k \to \tau_i$?

  In lemma we concluded that $\Gamma_i \ni x\colon \varphi_0 \wedge (\overline{\rho_1} \to \varphi_1 \wedge (\overline{\rho_2} \to \varphi_2 \wedge \ldots (\overline{\rho_k} \to \underline{\tau_i}) \ldots))$, and that's the x and k we need.

  All we need to do is to search in the set $\bigcup\limits_{(x\colon \varphi) \in \Gamma_j} unroll(\varphi)$ for the type of the form $\cdots \to \alpha$. Let's call the corresponding variable the candidate. Now we need to check that the candidate satisfies not only j-th requirement, but all the requirements. To do it we look at $unroll(\varphi)$, where $(x\colon \varphi_i) \in \Gamma_i$ for all i and check if there's a type with at least k arguments such that if we disconnect the first k arguments out of it, we get a tail $r \leqslant \tau_i$.

- $3^*_{\wedge}$.

  The difference with $3^*_{\wedge\eta}$ is in the part where we check that the candidate satisfies all the other requirements. Here we should preform $unroll(\varphi_i)$ more gently: we should stop it if the type we unroll has $\geq k$ arguments. And then we should check not $r \leqslant \tau_i$, but $r = \tau_i$.