HOT Compilation: Σ Kinds

TA: Akiva Leffert [*]

December 9, 2006

1 Introduction

In examining the singleton calculus in class, we found that we needed a function kind that was *dependent* in the sense that the codomain kind could depend upon the value the function was applied to, denoted $\Pi\alpha{:}k_1.\,k_2$. Recall that we saw this as a generalization of ordinary function kinds, writing $k_1 \to k_2$ for the degenerate non-dependent case where α does not appear free in k_2.

There is a similar generalization of pair kinds $k_1 \times k_2$, where the kind of the second component can depend on the first component. We write this dependent pair kind $\Sigma\alpha{:}k_1.\,k_2$. As before, we continue to write degenerate pair kinds as $k_1 \times k_2$, when α does not appear free in k_2.

(Somewhat confusingly, dependent pair kinds (Σ kinds) are often referred to as *dependent sums*, while dependent function kinds (Π kinds) are called *dependent products*. For this handout, we'll either call them dependent pairs and dependent functions or refer to them directly as Σ kinds and Π kinds.)

Dependent pairs are useful for understanding the type theoretic underpinnings of modular structure in languages like Standard ML. Standard ML's module language lets us create "translucent" signatures that expose or conceal varying degrees of information about the identities of types. For example, consider the signature:

```
sig
  type t
  type u = t -> t
end
```

In this signature, type `t`'s identity is completely hidden, while a certain amount of information is allowed to leak out about the identity of type `u`—namely that it is equivalent to `t -> t`, whatever `t` happens to be.

Dependent pair kinds along with singletons allow us to express similar dependency relationships among collections of constructors. The signature shown above might be represented by a pair kind like

$$\Sigma\alpha{:}\mathbf{T}.\,\mathcal{S}(\alpha \to \alpha),$$

which denotes the kind of a pair of constructors, the first having kind $\mathbf{T}$ and the second being definitionally equal to the function type from elements of the first type to elements of the first type.

In fact, in our elaborator IL, we will be representing modules and signatures using a generalization of dependent pairs to the n-ary case (while also exercising caution to distinguish labels like `t` from variables like α). It may be helpful, though, to keep the basic type theory of dependent pairs in mind as we explore the topic of elaboration. Σ kinds will arise more directly when we reach the topic of phase separation.

2 Technical details

The system we will consider in this handout is the dependently typed λ-calculus with functions, pairs, and singletons, called $\lambda_{\le}^{\Pi\Sigma S}$. We'll examine the main judgment forms of $\lambda_{\le}^{\Pi\Sigma S}$ and exhibit algorithms for

[*]Originally preparedy by William Lovas

$k ::= \mathbf{T}$	kind of base types
$\mid \mathcal{S}(c)$	singleton kind
$\mid \Pi\alpha{:}k_1.\, k_2$	kind of constructor functions
$\mid \Sigma\alpha{:}k_1.\, k_2$	kind of constructor pairs
$c ::= \alpha$	constructor variables
$\mid c_1 \to c_2$	function types
$\mid \forall\alpha{:}k.\, c$	polymorphic types
$\mid \lambda\alpha{:}k.\, c$	constructor functions
$\mid c_1\ c_2$	constructor application
$\mid \langle c_1, c_2\rangle$	constructor pairs
$\mid \pi_1\, c$	first projection
$\mid \pi_2\, c$	second projection
$\Gamma ::= \cdot$	empty context
$\mid \Gamma, \alpha{:}k$	kinding declaration

Figure 1: Syntax of $\lambda_{\leq}^{\Pi\Sigma S}$

$p ::= \alpha$	variables
$\mid p\ c$	application of a path
$\mid \pi_1\, p$	first projection from a path
$\mid \pi_2\, p$	second projection from a path

Figure 2: Syntax of constructor paths in $\lambda_{\leq}^{\Pi\Sigma S}$

deciding those judgments, building off the rules already presented in class. While a detailed examination of the metatheory of $\lambda_{\leq}^{\Pi\Sigma S}$, is beyond our scope, the interested reader may find further discussion in [1].

The syntax of $\lambda_{\leq}^{\Pi\Sigma S}$ is presented is Figures 1 and 2. Note that we've elided any discussion of expressions and expression typing; the only changes required to add dependent pair kinds to the language occur at the constructor and kind levels.

2.1 Declarative rules

The declarative judgments defining the static semantics of $\lambda_{\leq}^{\Pi\Sigma S}$ are shown in Figures 3 and 4. Most of the rules you've already seen in class; the new rules required for Σ kinds are displayed in shaded boxes. A few interesting rules have been named.

Both constructor kinding and constructor equivalence must respect subkinding, so both judgments have subsumption rules, K-Sub and Eq-Sub respectively. Both judgments also include extensionality principles at Π and Σ kinds; you may remember from class that extensionality is related to η-conversion: it permits us to reason about a constructor based on how it behaves.

Constructor equivalence also includes the usual β-conversion rules for both functions and pairs. Interestingly, these rules are in some sense "redundant": given extensional equivalence (rules Eq-Π-Ext and Eq-Σ-Ext) and equivalence at singleton kinds (rule Eq-Sing), the β rules are admissible! Further discussion of this point, along with examples and proofs, can be found in [1].

Recall that subkinding for Π kinds is contravariant in the domain and covariant in the codomain; subkinding for Σ kinds is covariant in both positions.

$$\boxed{\Gamma \vdash k : \mathbf{kind}}$$

$$\frac{}{\Gamma \vdash \mathbf{T} : \mathbf{kind}} \qquad \frac{\Gamma \vdash c : \mathbf{T}}{\Gamma \vdash \mathcal{S}(c) : \mathbf{kind}} \qquad \frac{\Gamma \vdash k_1 : \mathbf{kind} \qquad \Gamma, \alpha{:}k_1 \vdash k_2 : \mathbf{kind} \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \Pi\alpha{:}k_1.\, k_2 : \mathbf{kind}}$$

$$\frac{\Gamma \vdash k_1 : \mathbf{kind} \qquad \Gamma, \alpha{:}k_1 \vdash k_2 : \mathbf{kind} \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \Sigma\alpha{:}k_1.\, k_2 : \mathbf{kind}}$$

$$\boxed{\Gamma \vdash c : k}$$

$$\frac{\Gamma(\alpha) = k}{\Gamma \vdash \alpha : k} \qquad \frac{\Gamma \vdash c_1 : \mathbf{T} \qquad \Gamma \vdash c_2 : \mathbf{T}}{\Gamma \vdash c_1 \to c_2 : \mathbf{T}} \qquad \frac{\Gamma \vdash k : \mathbf{kind} \qquad \Gamma, \alpha{:}k \vdash c : \mathbf{T} \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \forall\alpha{:}k.\, c : \mathbf{T}}$$

$$\frac{\Gamma \vdash k : \mathbf{kind} \qquad \Gamma, \alpha{:}k \vdash c : k' \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \lambda\alpha{:}k.\, c : \Pi\alpha{:}k.\, k'} \qquad \frac{\Gamma \vdash c_1 : \Pi\alpha{:}k.\, k' \qquad \Gamma \vdash c_2 : k}{\Gamma \vdash c_1\ c_2 : [c_2/\alpha]k'}$$

$$\frac{\Gamma \vdash c_1 : k_1 \qquad \Gamma \vdash c_2 : [c_1/\alpha]k_2 \qquad \Gamma \vdash \Sigma\alpha{:}k_1.\, k_2 : \mathbf{kind}}{\Gamma \vdash \langle c_1, c_2\rangle : \Sigma\alpha{:}k_1.\, k_2} \qquad \frac{\Gamma \vdash c : \Sigma\alpha{:}k_1.\, k_2}{\Gamma \vdash \pi_1\, c : k_1} \qquad \frac{\Gamma \vdash c : \Sigma\alpha{:}k_1.\, k_2}{\Gamma \vdash \pi_2\, c : [\pi_1\, c/\alpha]k_2}$$

$$\frac{\Gamma \vdash c \equiv c' : \mathbf{T}}{\Gamma \vdash c : \mathcal{S}(c')}\ (\text{K-Sing-Intro}) \qquad \frac{\Gamma \vdash c : k' \qquad \Gamma \vdash k' \leq k}{\Gamma \vdash c : k}\ (\text{K-Sub})$$

$$\frac{\Gamma \vdash k : \mathbf{kind} \qquad \Gamma, \alpha{:}k \vdash c\ \alpha : k' \qquad \Gamma \vdash c : \Pi\alpha{:}k.\, k'' \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash c : \Pi\alpha{:}k.\, k'}\ (\text{K-}\Pi\text{-Ext})$$

$$\frac{\Gamma \vdash \pi_1\, c : k_1 \qquad \Gamma \vdash \pi_2\, c : [\pi_1\, c/\alpha]k_2 \qquad \Gamma \vdash \Sigma\alpha{:}k_1.\, k_2 : \mathbf{kind}}{\Gamma \vdash c : \Sigma\alpha{:}k_1.\, k_2}\ (\text{K-}\Sigma\text{-Ext})$$

$$\boxed{\Gamma \vdash k_1 \leq k_2}$$

$$\frac{}{\Gamma \vdash \mathbf{T} \leq \mathbf{T}} \qquad \frac{\Gamma \vdash c : \mathbf{T}}{\Gamma \vdash \mathcal{S}(c) \leq \mathbf{T}}\ (\text{S-Sing}) \qquad \frac{\Gamma \vdash c_1 \equiv c_2 : \mathbf{T}}{\Gamma \vdash \mathcal{S}(c_1) \leq \mathcal{S}(c_2)}$$

$$\frac{\Gamma \vdash k_2 \leq k_1 \qquad \Gamma, \alpha{:}k_2 \vdash k_1' \leq k_2' \qquad \Gamma, \alpha{:}k_1 \vdash k_1' : \mathbf{kind} \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \Pi\alpha{:}k_1.\, k_1' \leq \Pi\alpha{:}k_2.\, k_2'}\ (\text{S-}\Pi)$$

$$\frac{\Gamma \vdash k_1 \leq k_2 \qquad \Gamma, \alpha{:}k_1 \vdash k_1' \leq k_2' \qquad \Gamma, \alpha{:}k_2 \vdash k_2' : \mathbf{kind} \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \Sigma\alpha{:}k_1.\, k_1' \leq \Sigma\alpha{:}k_2.\, k_2'}\ (\text{S-}\Sigma)$$

Figure 3: Declarative judgments: kind formation, kinding, and subkinding

$$\boxed{\Gamma \vdash c_1 \equiv c_2 : k}$$

$$\frac{\Gamma \vdash c : k}{\Gamma \vdash c \equiv c : k} \qquad \frac{\Gamma \vdash c_1 \equiv c_2 : k}{\Gamma \vdash c_2 \equiv c_1 : k} \qquad \frac{\Gamma \vdash c_1 \equiv c_2 : k \qquad \Gamma \vdash c_2 \equiv c_3 : k}{\Gamma \vdash c_1 \equiv c_3 : k}$$

$$\frac{\Gamma(\alpha) = k}{\Gamma \vdash \alpha \equiv \alpha : k} \qquad \frac{\Gamma \vdash c_1 \equiv c_2 : \mathbf{T} \qquad \Gamma \vdash c_1' \equiv c_2' : \mathbf{T}}{\Gamma \vdash c_1 \to c_1' \equiv c_2 \to c_2' : \mathbf{T}}$$

$$\frac{\Gamma \vdash k : \mathbf{kind} \qquad \Gamma, \alpha{:}k \vdash c_1 \equiv c_2 : \mathbf{T} \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \forall \alpha{:}k.\, c_1 \equiv \forall \alpha{:}k.\, c_2 : \mathbf{T}}$$

$$\frac{\Gamma \vdash k : \mathbf{kind} \qquad \Gamma, \alpha{:}k \vdash c_1 \equiv c_2 : k' \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \lambda \alpha{:}k.\, c_1 \equiv \lambda \alpha{:}k.\, c_2 : \Pi \alpha{:}k.\, k'} \qquad \frac{\Gamma \vdash c_1 \equiv c_2 : \Pi \alpha{:}k.\, k' \qquad \Gamma \vdash c_1' \equiv c_2' : k}{\Gamma \vdash c_1\ c_1' \equiv c_2\ c_2' : [c_1'/\alpha]k'}$$

$$\frac{\Gamma \vdash c_1 \equiv c_1' : k_1 \qquad \Gamma \vdash c_2 \equiv c_2' : [c_1/\alpha]k_2 \qquad \Gamma \vdash \Sigma \alpha{:}k_1.\, k_2 : \mathbf{kind}}{\Gamma \vdash \langle c_1, c_2 \rangle \equiv \langle c_1', c_2' \rangle : \Sigma \alpha{:}k_1.\, k_2}$$

$$\frac{\Gamma \vdash c \equiv c' : \Sigma \alpha{:}k_1.\, k_2}{\Gamma \vdash \pi_1\, c \equiv \pi_1\, c' : k_1} \qquad \frac{\Gamma \vdash c \equiv c' : \Sigma \alpha{:}k_1.\, k_2}{\Gamma \vdash \pi_2\, c \equiv \pi_2\, c' : [\pi_1\, c/\alpha]k_2}$$

$$\frac{\Gamma \vdash c : \mathcal{S}(c')}{\Gamma \vdash c \equiv c' : \mathcal{S}(c')}\ (\textsc{Eq-Sing}) \qquad \frac{\Gamma \vdash c : \mathcal{S}(c')}{\Gamma \vdash c \equiv c' : \mathbf{T}}\ (\textsc{Eq-Sing-Elim})$$

$$\frac{\Gamma \vdash c_1 \equiv c_2 : k \qquad \Gamma \vdash k \leq k'}{\Gamma \vdash c_1 \equiv c_2 : k'}\ (\textsc{Eq-Sub})$$

$$\frac{\Gamma, \alpha{:}k \vdash c_1 : k' \qquad \Gamma \vdash c_2 : k}{\Gamma \vdash (\lambda \alpha{:}k.\, c_1)\ c_2 \equiv [c_2/\alpha]c_1 : [c_2/\alpha]k'}\ (\textsc{Eq-}\Pi\textsc{-}\beta)$$

$$\frac{\Gamma \vdash c_1 : k_1 \qquad \Gamma \vdash c_2 : k_2}{\Gamma \vdash \pi_1\, \langle c_1, c_2 \rangle \equiv c_1 : k_1}\ (\textsc{Eq-}\Sigma\textsc{-}\beta_1) \qquad \frac{\Gamma \vdash c_1 : k_1 \qquad \Gamma \vdash c_2 : k_2}{\Gamma \vdash \pi_2\, \langle c_1, c_2 \rangle \equiv c_2 : k_2}\ (\textsc{Eq-}\Sigma\textsc{-}\beta_2)$$

$$\frac{\begin{array}{c}\Gamma \vdash k : \mathbf{kind} \qquad \Gamma, \alpha{:}k \vdash c_1\ \alpha \equiv c_2\ \alpha : k' \qquad (\alpha \notin \mathrm{dom}(\Gamma)) \\ \Gamma \vdash c_1 : \Pi \alpha{:}k.\, k'' \qquad \Gamma \vdash c_2 : \Pi \alpha{:}k.\, k''' \end{array}}{\Gamma \vdash c_1 \equiv c_2 : \Pi \alpha{:}k.\, k'}\ (\textsc{Eq-}\Pi\textsc{-Ext})$$

$$\frac{\Gamma \vdash \pi_1\, c \equiv \pi_1\, c' : k_1 \qquad \Gamma \vdash \pi_2\, c \equiv \pi_2\, c' : [\pi_1\, c/\alpha]k_2 \qquad \Gamma \vdash \Sigma \alpha{:}k_1.\, k_2 : \mathbf{kind}}{\Gamma \vdash c \equiv c' : \Sigma \alpha{:}k_1.\, k_2}\ (\textsc{Eq-}\Sigma\textsc{-Ext})$$

Figure 4: Declarative judgments, continued: constructor equivalence

$$\begin{aligned}
\mathcal{S}_{\mathbf{T}}(c) &\overset{\text{def}}{=} \mathcal{S}(c) \\
\mathcal{S}_{\mathcal{S}(c')}(c) &\overset{\text{def}}{=} \mathcal{S}(c) \\
\mathcal{S}_{\Pi\alpha:k_1.\,k_2}(c) &\overset{\text{def}}{=} \Pi\alpha{:}k_1.\,\mathcal{S}_{k_2}(c\ \alpha) && \text{(where } \alpha \notin FV(c)) \\
\mathcal{S}_{\Sigma\alpha:k_1.\,k_2}(c) &\overset{\text{def}}{=} \mathcal{S}_{k_1}(\pi_1\, c) \times \mathcal{S}_{[\pi_1\, c/\alpha]k_2}(\pi_2\, c)
\end{aligned}$$

Figure 5: Singletons at higher kinds

2.2 Algorithmic rules

$\lambda_{\leq}^{\Pi\Sigma S}$ enjoys both decidable kind-checking and decidable constructor equivalence (though the proofs of these facts are decidedly non-trivial). Algorithmic rules for these judgments and judgments they depend upon appear in Figures 6, 7, and 8. Additionally, we must extend our definition of higher-order singletons to account for singletons of constructors with Σ kinds; this is done in Figure 5.

Figure 6 shows the judgments most closely related to constructor kinding: kind formation, kind checking, kind synthesis, and subkinding. The subsumption rule K-Sub has been replaced with the kind-checking rule AK-Sub which invokes algorithmic subkinding; kind checking is used anytime a constructor is required to have a particular kind, as in the premise for the argument in the rule for constructor application.

The rules for algorithmic subkinding are nearly identical to the declarative ones with all declarative judgments replaced by the corresponding algorithmic ones. The main outlier is the singleton rule, AS-Sing, which lacks a premise corresponding to S-Sing's premise checking that the constructor c is well-formed at kind $\mathbf{T}$. This premise can be omitted in the algorithmic rule because we presuppose that all the inputs of any algorithmic judgment are well-formed in the appropriate context; since $\mathcal{S}(c)$ is well-formed only if c has kind $\mathbf{T}$, we needn't check this fact. Similarly, AS-Π and AS-Σ needn't include premises to ensure the well-formedness of both kinds in the conclusion like the declarative rules S-Π and S-Σ must.

It is interesting to note that the principal kind of a constructor pair is never a dependent pair kind, but rather always a degenerate $\times$ kind. Similarly, our definition of singletons at Σ kinds yields a degenerate non-dependent pair kind[1]. One might be surprised at this, since dependency was absolutely essential in defining singletons at function kinds—indeed, once we added singletons to the language, we *had* to add dependent functions!

Intuitively, the same situation can be sidestepped with regard to pair kinds because we always know both components of a constructor pair c statically: the first component is $\pi_1\, c$ and the second is $\pi_2\, c$. Since we have the ability to name both components, we can eliminate any dependency in the second component's kind simply by substituting $\pi_1\, c$ for the Σ-bound variable. No similar trick exists for dependent function kinds since the argument to the function is fundamentally unknown: until we apply the function, its argument can be named only by the Π-bound variable.

Weak head normalization and its constituent judgments are defined in Figure 7. We extend natural kind extraction to Σ kinds in the obvious way, taking care to eliminate dependencies from the kind of a second projection. Weak head reduction is supplemented with the usual β and congruence rules for constructor pairs.

The equivalence algorithm in Figure 8 is the same kind-directed algorithm we've seen before: the extensionality principles serve to drive comparison down to base kind $\mathbf{T}$, and then constructors are weak head normalized and compared structurally. Recall that we presuppose the well-formedness of the inputs, so the rule for comparison at singleton kinds simply succeeds: if both constructors have the same singleton kind, they must be equivalent.

[1]If we wanted more uniformity, we could have defined things differently to keep Σs around in principal pair kinds and singletons at pair kinds, but eliminating the dependency makes our implementation burden a bit lighter.

$$\boxed{\Gamma \vdash k \Leftarrow \mathbf{kind}}$$

$$\frac{}{\Gamma \vdash \mathbf{T} \Leftarrow \mathbf{kind}} \qquad \frac{\Gamma \vdash c \Leftarrow \mathbf{T}}{\Gamma \vdash \mathcal{S}(c) \Leftarrow \mathbf{kind}} \qquad \frac{\Gamma \vdash k_1 \Leftarrow \mathbf{kind} \qquad \Gamma, \alpha{:}k_1 \vdash k_2 \Leftarrow \mathbf{kind} \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \Pi\alpha{:}k_1.\, k_2 \Leftarrow \mathbf{kind}}$$

$$\frac{\Gamma \vdash k_1 \Leftarrow \mathbf{kind} \qquad \Gamma, \alpha{:}k_1 \vdash k_2 \Leftarrow \mathbf{kind} \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \Sigma\alpha{:}k_1.\, k_2 \Leftarrow \mathbf{kind}}$$

$$\boxed{\Gamma \vdash c \Leftarrow k}$$

$$\frac{\Gamma \vdash c \Rightarrow k' \qquad \Gamma \vdash k' \trianglelefteq k}{\Gamma \vdash c \Leftarrow k} \text{ (AK-Sub)}$$

$$\boxed{\Gamma \vdash c \Rightarrow k}$$

$$\frac{\Gamma(\alpha) = k}{\Gamma \vdash \alpha \Rightarrow \mathcal{S}_k(\alpha)} \qquad \frac{\Gamma \vdash c_1 \Leftarrow \mathbf{T} \qquad \Gamma \vdash c_2 \Leftarrow \mathbf{T}}{\Gamma \vdash c_1 \to c_2 \Rightarrow \mathbf{T}} \qquad \frac{\Gamma \vdash k \Leftarrow \mathbf{kind} \qquad \Gamma, \alpha{:}k \vdash c \Leftarrow \mathbf{T} \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \forall\alpha{:}k.\, c \Rightarrow \mathbf{T}}$$

$$\frac{\Gamma \vdash k \Leftarrow \mathbf{kind} \qquad \Gamma, \alpha{:}k \vdash c \Rightarrow k' \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \lambda\alpha{:}k.\, c \Rightarrow \Pi\alpha{:}k.\, k'} \qquad \frac{\Gamma \vdash c_1 \Rightarrow \Pi\alpha{:}k.\, k' \qquad \Gamma \vdash c_2 \Leftarrow k}{\Gamma \vdash c_1\ c_2 \Rightarrow [c_2/\alpha]k'}$$

$$\frac{\Gamma \vdash c_1 \Rightarrow k_1 \qquad \Gamma \vdash c_2 \Rightarrow k_2}{\Gamma \vdash \langle c_1, c_2 \rangle \Rightarrow k_1 \times k_2} \qquad \frac{\Gamma \vdash c \Rightarrow k_1 \times k_2}{\Gamma \vdash \pi_1\, c \Rightarrow k_1} \qquad \frac{\Gamma \vdash c \Rightarrow k_1 \times k_2}{\Gamma \vdash \pi_2\, c \Rightarrow k_2}$$

$$\boxed{\Gamma \vdash k_1 \trianglelefteq k_2}$$

$$\frac{}{\Gamma \vdash \mathbf{T} \trianglelefteq \mathbf{T}} \qquad \frac{}{\Gamma \vdash \mathcal{S}(c) \trianglelefteq \mathbf{T}} \text{ (AS-Sing)} \qquad \frac{\Gamma \vdash c_1 \Leftrightarrow c_2 : \mathbf{T}}{\Gamma \vdash \mathcal{S}(c_1) \trianglelefteq \mathcal{S}(c_2)}$$

$$\frac{\Gamma \vdash k_2 \trianglelefteq k_1 \qquad \Gamma, \alpha{:}k_2 \vdash k_1' \trianglelefteq k_2' \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \Pi\alpha{:}k_1.\, k_1' \trianglelefteq \Pi\alpha{:}k_2.\, k_2'} \text{ (AS-}\Pi\text{)}$$

$$\frac{\Gamma \vdash k_1 \trianglelefteq k_2 \qquad \Gamma, \alpha{:}k_1 \vdash k_1' \trianglelefteq k_2' \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \Sigma\alpha{:}k_1.\, k_1' \trianglelefteq \Sigma\alpha{:}k_2.\, k_2'} \text{ (AS-}\Sigma\text{)}$$

Figure 6: Algorithmic judgments: kind formation, kind checking/principal kind synthesis, and subkind checking

$$\boxed{\Gamma \vdash p \uparrow k}$$

$$\frac{\Gamma(\alpha) = k}{\Gamma \vdash \alpha \uparrow k} \qquad \frac{\Gamma \vdash p \uparrow \Pi\alpha{:}k.\, k'}{\Gamma \vdash p\ c \uparrow [c/\alpha]k'} \qquad \frac{\Gamma \vdash p \uparrow \Sigma\alpha{:}k_1.\, k_2}{\Gamma \vdash \pi_1\, p \uparrow k_1} \qquad \frac{\Gamma \vdash p \uparrow \Sigma\alpha{:}k_1.\, k_2}{\Gamma \vdash \pi_2\, p \uparrow [\pi_1\, p/\alpha]k_2}$$

$$\boxed{\Gamma \vdash c_1 \rightsquigarrow c_2}$$

$$\frac{}{\Gamma \vdash (\lambda\alpha{:}k.\, c_1)\ c_2 \rightsquigarrow [c_2/\alpha]c_1}\ (\text{R-}\Pi\text{-}\beta) \qquad \frac{\Gamma \vdash c_1 \rightsquigarrow c_1'}{\Gamma \vdash c_1\ c_2 \rightsquigarrow c_1'\ c_2}$$

$$\frac{}{\Gamma \vdash \pi_1\, \langle c_1, c_2 \rangle \rightsquigarrow c_1}\ (\text{R-}\Sigma\text{-}\beta_1) \qquad \frac{}{\Gamma \vdash \pi_2\, \langle c_1, c_2 \rangle \rightsquigarrow c_2}\ (\text{R-}\Sigma\text{-}\beta_2)$$

$$\frac{\Gamma \vdash c \rightsquigarrow c'}{\Gamma \vdash \pi_1\, c \rightsquigarrow \pi_1\, c'} \qquad \frac{\Gamma \vdash c \rightsquigarrow c'}{\Gamma \vdash \pi_2\, c \rightsquigarrow \pi_2\, c'}$$

$$\frac{\Gamma \vdash p \uparrow \mathcal{S}(c)}{\Gamma \vdash p \rightsquigarrow c}$$

$$\boxed{\Gamma \vdash c_1 \Downarrow c_2}$$

$$\frac{\Gamma \vdash c \not\rightsquigarrow}{\Gamma \vdash c \Downarrow c} \qquad \frac{\Gamma \vdash c \rightsquigarrow c' \qquad \Gamma \vdash c' \Downarrow c''}{\Gamma \vdash c \Downarrow c''}$$

Figure 7: Algorithmic judgments, continued: natural kind extraction, weak head reduction, and weak head normalization

$$\boxed{\Gamma \vdash c_1 \Leftrightarrow c_2 : k}$$

$$\frac{}{\Gamma \vdash c_1 \Leftrightarrow c_2 : \mathcal{S}(c)} \qquad \frac{\Gamma \vdash c_1 \Downarrow c_1' \qquad \Gamma \vdash c_2 \Downarrow c_2' \qquad \Gamma \vdash c_1' \leftrightarrow c_2' : k}{\Gamma \vdash c_1 \Leftrightarrow c_2 : \mathbf{T}}$$

$$\frac{\Gamma, \alpha{:}k \vdash c_1\ \alpha \Leftrightarrow c_2\ \alpha : k' \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash c_1 \Leftrightarrow c_2 : \Pi\alpha{:}k.\, k'} \ (\text{AEq-}\Pi\text{-Ext})$$

$$\frac{\Gamma \vdash \pi_1\, c \Leftrightarrow \pi_1\, c' : k_1 \qquad \Gamma \vdash \pi_2\, c \Leftrightarrow \pi_2\, c' : [\pi_1\, c/\alpha]k_2}{\Gamma \vdash c \Leftrightarrow c' : \Sigma\alpha{:}k_1.\, k_2} \ (\text{AEq-}\Sigma\text{-Ext})$$

$$\boxed{\Gamma \vdash c_1 \leftrightarrow c_2 : k}$$

$$\frac{\Gamma(\alpha) = k}{\Gamma \vdash \alpha \leftrightarrow \alpha : k} \qquad \frac{\Gamma \vdash c_1 \Leftrightarrow c_2 : \mathbf{T} \qquad \Gamma \vdash c_1' \Leftrightarrow c_2' : \mathbf{T}}{\Gamma \vdash (c_1 \to c_1') \leftrightarrow (c_2 \to c_2') : \mathbf{T}} \qquad \frac{\Gamma, \alpha{:}k \vdash c_1 \Leftrightarrow c_2 : \mathbf{T} \qquad (\alpha \notin \mathrm{dom}(\Gamma))}{\Gamma \vdash \forall\alpha{:}k.\, c_1 \leftrightarrow \forall\alpha{:}k.\, c_2 : \mathbf{T}}$$

$$\frac{\Gamma \vdash c_1 \leftrightarrow c_2 : \Pi\alpha{:}k.\, k' \qquad \Gamma \vdash c_1' \Leftrightarrow c_2' : k}{\Gamma \vdash c_1\ c_1' \leftrightarrow c_2\ c_2' : [c_1'/\alpha]k'} \qquad \frac{\Gamma \vdash c \leftrightarrow c' : \Sigma\alpha{:}k_1.\, k_2}{\Gamma \vdash \pi_1\, c \leftrightarrow \pi_1\, c' : k_1} \qquad \frac{\Gamma \vdash c \leftrightarrow c' : \Sigma\alpha{:}k_1.\, k_2}{\Gamma \vdash \pi_2\, c \leftrightarrow \pi_2\, c' : [\pi_1\, c/\alpha]k_2}$$

Figure 8: Algorithmic judgments, continued: kind-directed equivalence and structural equivalence

References

[1] Christopher A. Stone and Robert Harper. Extensional equivalence and singleton types. *ACM Transactions on Computational Logic*, to appear. Available electronically at `http://www.cs.hmc.edu/~stone/papers/tocl-final.pdf`.