DECIDING CONSTRUCTOR EQUIVALENCE

Abstract. In the previous section we described definitional equality for constructors. The presentation given the most natural definition of the operator $\equiv$, but is not so amenable to translation to code, which will complicate the implementation of type-checking. Here we will describe a couple of algorithms to decide whether $\Gamma \vdash c_1 \equiv c_2 : k$. The first, **normalize-and-compare** is conceptually appealing but doesn't scale to some richer calculi we will cover. So we will develop a second approach, called **algorithmic constructor equivalence**, which is an inductively defined judgement $\Gamma \vdash c_1 \iff c_2 : k$ that better lends itself to implementation in code.

1. Normalize-and-compare

Recall the syntactic specification of (souped-up) F^ω.

$$\begin{array}{lcl} k & ::= & \texttt{type} \mid k \to k \mid k * k \\ c & ::= & \alpha \mid c \to c \mid \forall\alpha : k.c \mid \lambda\alpha : k.c \mid c\ c \mid \langle c, c\rangle \mid \pi_1\ c \mid \pi_2\ c \\ e & ::= & x \mid \lambda x : c.e \mid e\ e \mid \Lambda\alpha : k.e \mid e[c] \\ \Gamma & ::= & \cdot \mid \Gamma, x : c \mid \Gamma, \alpha : k \end{array}$$

The idea of normalize-and-compare is simple. We would give a pair of inductively defined judgements, $\Gamma \vdash c\ \mathsf{normal}$ and $\Gamma \vdash c \rightsquigarrow c'$, which would be defined such that if $\vdash c : k$ (that is, c is well-formed) then c would eventually step $\rightsquigarrow$ to some c' with $c'\ \mathsf{normal}$ - the idea being that normalized constructors are easy to check for equality. Examples of the rules defining these judgements include

$$\text{1A}\quad \frac{}{\Gamma, \alpha : \texttt{type} \vdash \alpha\ \mathsf{normal}} \qquad \text{1B}\quad \frac{\Gamma \vdash c_1\ \mathsf{normal} \quad \Gamma \vdash c_2\ \mathsf{normal}}{\langle c_1, c_2\rangle\ \mathsf{normal}}$$

$$\text{1C}\quad \frac{\Gamma, \alpha : k \vdash c\ \mathsf{normal}}{\Gamma \vdash (\forall\alpha : k.\ c)\ \mathsf{normal}} \qquad \text{1D}\quad \frac{\Gamma \vdash c \rightsquigarrow c'}{\Gamma \vdash \pi_1\ c \rightsquigarrow \pi_1\ c'} \qquad \text{1E}\quad \frac{\Gamma \vdash c_2\ \mathsf{normal} \quad \Gamma \vdash c_2 : k}{\Gamma \vdash (\lambda\alpha : k.\ c_1)\ c_2 \rightsquigarrow [c_2/\alpha]c_1}$$

And so forth. Additionally, we specify a judgement for the transitive closure of $\rightsquigarrow$:

$$\text{1F}\quad \frac{c \rightsquigarrow c' \quad c' \rightsquigarrow^* c''}{c \rightsquigarrow^* c''} \qquad \text{1G}\quad \frac{c\ \mathsf{normal}}{c \rightsquigarrow^* c}$$

at last we would develop a judgement $\Gamma \vdash c_1 \equiv_n c_2 : k$, which would be a simple structural equivalence comparison on normalized constructors. Then the algorithm to check $\Gamma \vdash c_1 \equiv c_2 : k$ is as follows.

(1) Determine whether $\vdash c_1 : k_1$ and $\vdash c_2 : k_2$ - that is, that c_1 and c_2 are well-formed constructors of some kinds k_1 and k_2 respectively.
(2) If so, check whether k_1 is k_2; this is a simple comparison because we don't have a complicated kind structure.
(3) If so, compute c_1' and c_2' such that $c_1 \rightsquigarrow^* c_1'$ and $c_2 \rightsquigarrow^* c_2'$.

(4) Determine whether $\Gamma \vdash c_1' \equiv_n c_2' : k$.

The reader may supply the remaining rules. We are going to choose to focus on a different algorithm however, because eventually we will cover the "singleton-kind calculus," which doesn't play nice with normalize-and-compare.

2. Algorithmic Constructor Equivalence

The goal now is define algorithmic constructor equivalence, which is specified by the judgement $\Gamma \vdash c_1 \iff c_2 : k$. In order to define it we will use a number of auxiliary judgements, which are summarized in the table below. I annotated the judgements with polarities + and −, which indicate whether the corresponding variable should be an input or an output, respectively, when the judgement is implemented in code. Implementation of judgements which have no − variables simply determine whether the judgement is derivable.

Judgement	Description
$\Gamma^+ \vdash c_1^+ \iff c_2^+ : k^+$	Algorithmic constructor equivalence
$\Gamma^+ \vdash c_1^+ \longleftrightarrow c_2^+ : k^-$	Algorithmic path equivalence
$c^+ \Downarrow n^-$	Weak-head normalization
$c^+ \rightsquigarrow c'^-$	Weak-head reduction
$\Gamma^+ \vdash e^+ \Rightarrow k^-$	Kind synthesis
$\Gamma^+ \vdash e^+ \Leftarrow k^+$	Kind checking
$\Gamma^+ \vdash e^+ \Rightarrow \tau^-$	Type synthesis
$\Gamma^+ \vdash e^+ \Leftarrow \tau^+$	Type checking