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.

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

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

1A
$$\frac{}{\Gamma, \alpha : \mathtt{type} \vdash \alpha\ \mathsf{normal}}$$

1B
$$\frac{\Gamma \vdash c_1\ \mathsf{normal} \quad \Gamma \vdash c_2\ \mathsf{normal}}{\Gamma \vdash \langle c_1, c_2 \rangle\ \mathsf{normal}}$$

1C
$$\frac{\Gamma, \alpha : k \vdash c\ \mathsf{normal}}{\Gamma \vdash (\forall \alpha : k.\ c)\ \mathsf{normal}}$$

1D
$$\frac{\Gamma \vdash c \rightsquigarrow c'}{\Gamma \vdash \pi_1\ c \rightsquigarrow \pi_1\ c'}$$

1E
$$\frac{\Gamma \vdash c_2\ \mathsf{normal} \quad \Gamma \vdash c_2 : k}{\Gamma \vdash (\lambda \alpha : k.\ c_1)\ c_2 \rightsquigarrow [c_2/\alpha]c_1}$$

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

1F
$$\frac{c \rightsquigarrow c' \quad c' \rightsquigarrow^* c''}{c \rightsquigarrow^* c''}$$

1G
$$\frac{c\ \mathsf{normal}}{c \rightsquigarrow^* c}$$

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

The idea of this algorithm is similar to normalize-and-compare, but it is based on "weak-head normalization" rather than plain normalization. The difference will become clear shortly. The principal rules defining the judgement $\Gamma \vdash c_1 \iff c_2 : k$ are

$$\text{2A}\ \frac{\Gamma, \alpha : k_1 \vdash c\ \alpha \iff c'\ \alpha : k_2}{\Gamma \vdash c \iff c' : k_1 \to k_2}$$

$$\text{2B}\ \frac{\Gamma \vdash \pi_1\ c \iff \pi_1\ c' : k_1 \quad \Gamma \vdash \pi_2\ c \iff \pi_2\ c' : k_2}{\Gamma \vdash c \iff c' : k_1 * k_2}$$

$$\text{2C}\ \frac{c_1 \Downarrow n_1 \quad c_2 \Downarrow n_2 \quad \Gamma \vdash n_1 \longleftrightarrow n_2 : \texttt{type}}{\Gamma \vdash c_1 \iff c_2 : \texttt{type}}$$

2c is the most interesting rule here. We will define a judgement $\Gamma \vdash n_1 \longleftrightarrow n_2 : k$ which accepts weak-head normalized versions of given constructors and uses them to decide equivalence. The judgement $c \Downarrow n$ specifies that n is the weak-head normalization of c. We specify it now.

$$\text{2D}\ \frac{c \rightsquigarrow c' \quad c' \Downarrow c''}{c \Downarrow c''} \qquad \text{2E}\ \frac{c \not\rightsquigarrow}{c \Downarrow c} \qquad \text{2F}\ \frac{}{(\lambda\alpha : k.\ c_1)c_2 \rightsquigarrow [c_2/\alpha]c_1} \qquad \text{2G}\ \frac{}{\pi_1\ \langle c_1, c_2\rangle \rightsquigarrow c_1}$$

$$\text{2H}\ \frac{}{\pi_2\ \langle c_1, c_2\rangle \rightsquigarrow c_2} \qquad \text{2I}\ \frac{c_1 \rightsquigarrow c_1'}{c_1\ c_2 \rightsquigarrow c_1'\ c_2}$$

Once weak-head normalization is performed, the syntactic range of constructors we need to consider is significantly reduced. Particularly, if paths are "defined" as follows:

$$p \quad ::= \quad \alpha \mid p\ c \mid \pi_1\ c \mid \pi_2\ c$$

Then a weak-head normalized constructor has the form

$$n \quad ::= \quad p \mid c_1 \to c_2 \mid \forall \alpha : k.\ c$$

To get an intuition for what's going on here, the thing to understand is that paths have the form $\alpha\ c_1 \ldots c_k$; that is, the outermost constructor is evaluated as far as possible. Comparing paths for equivalence is "easy," particularly because if the outermost constructors differ no further work needs to be done. We can now define algorithmic path equivalence, $\Gamma \vdash n \longleftrightarrow n' : k$. Notice that it's definition is mutually recursive with $\Gamma \vdash c \iff c' : k$. The rules are defined so that algorithmic path equivalence works as intended when given constructors in weak-head normal form.

2J
$$\frac{\alpha : k \in \Gamma}{\Gamma \vdash \alpha \longleftrightarrow \alpha : k}$$

2K
$$\frac{\Gamma \vdash p \longleftrightarrow p' : k_1 \to k_2 \quad \Gamma \vdash c \iff c' : k_1}{\Gamma \vdash p\ c \longleftrightarrow p'\ c' : k_2}$$

2L
$$\frac{\Gamma \vdash p \longleftrightarrow p' : k_1 * k_2}{\Gamma \vdash \pi_1\ p \longleftrightarrow \pi_1\ p' : k_1}$$

2M
$$\frac{\Gamma \vdash p \longleftrightarrow p' : k_1 * k_2}{\Gamma \vdash \pi_2\ p \longleftrightarrow \pi_2\ p' : k_2}$$

2N
$$\frac{\Gamma \vdash c_1 \iff c_1' : \texttt{type} \quad \Gamma \vdash c_2 \iff c_2' : \texttt{type}}{\Gamma \vdash (c_1 \to c_2) \longleftrightarrow (c_1' \to c_2') : \texttt{type}}$$

2O
$$\frac{\Gamma, \alpha : k \vdash c \iff c' : \texttt{type}}{\Gamma \vdash (\forall \alpha : k.\ c) \longleftrightarrow (\forall \alpha : k.\ c') : \texttt{type}}$$

Then, at last, we can state the following critical theorems:

(1) The definition of $\Gamma \vdash c_1 \iff c_2 : k$ is inductive - in particular, proof search for a given $\Gamma \vdash c_2 \iff c_2 : k$ terminates. This is not true for $\Gamma \vdash c_1 \equiv c_2 : k$!
(2) (Soundness) If Γ is well-formed, $\Gamma \vdash c_1 : k, \Gamma \vdash c_2 : k$, and $\Gamma \vdash c_1 \iff c_2 : k$, then $\Gamma \vdash c_1 \equiv c_2 : k$.
(3) (Completeness) If Γ is well-formed, $\Gamma \vdash c_1 : k, \Gamma \vdash c_2 : k$ and $\Gamma \vdash c_1 \equiv c_2 : k$, then $\Gamma \vdash c_1 \iff c_2 : k$.

At this point, you may be wondering, "If the whole point of weak-head normalization is to get rid of function applications, why do we have an algorithmic path equivalence rule for $p\ c \longleftrightarrow p'\ c'$?" It would seem that this case of a path applied to a constructor would never come up.

As it turns out, though, this is an important rule. When we weak-head reduce, there is no rule that applies to the case of variables. So we could have a weak-head normal form that looks like $\alpha\ c$, for some α.

An example of when such a case might occur is when attempting to derive $\alpha : \texttt{type} \to \texttt{type} \vdash \alpha \iff \alpha$. A good exercise is to see which rules apply to derive this statement.

3. Type checking and kind checking

The typing and kinding rules are more straightforward than constructor equivalence. The judgements we will define are:

Judgement	Description
$\Gamma^+ \vdash e^+ \Rightarrow k^-$	Kind synthesis
$\Gamma^+ \vdash e^+ \Leftarrow k^+$	Kind checking
$\Gamma^+ \vdash e^+ \Rightarrow \tau^-$	Type synthesis
$\Gamma^+ \vdash e^+ \Leftarrow \tau^+$	Type checking

The rules for kinds are:

3A
$$\frac{\alpha : k \in \Gamma}{\Gamma \vdash \alpha \Rightarrow k}$$

3B
$$\frac{\Gamma, \alpha : k \vdash c \Rightarrow k'}{\Gamma \vdash \lambda \alpha : k.\ c \Rightarrow k \to k'}$$

3C
$$\frac{\Gamma \vdash c_1 \Rightarrow k \to k' \quad \Gamma \vdash c_2 \Leftarrow k}{\Gamma \vdash c_1 \to c_2 \Rightarrow k'}$$

3D
$$\frac{\Gamma \vdash c \Rightarrow k}{\Gamma \vdash c \Leftarrow k}$$

3E
$$\frac{\Gamma \vdash c_1 \Rightarrow k_1 \quad \Gamma \vdash c_2 \Rightarrow k_2}{\Gamma \vdash \langle c_1, c_2 \rangle \Rightarrow k_1 * k_2}$$

3F
$$\frac{\Gamma \vdash c_1 \Leftarrow \texttt{type} \quad \Gamma \vdash c_2 \Leftarrow \texttt{type}}{\Gamma \vdash c_1 \to c_2 \Rightarrow \texttt{type}}$$

3G
$$\frac{\Gamma, \alpha : k \vdash c \Leftarrow \texttt{type}}{\Gamma \vdash \forall \alpha : k.\ c \Rightarrow \texttt{type}}$$

For types:

3H
$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x \Rightarrow \tau}$$

3I
$$\frac{\Gamma \vdash \tau \Leftarrow \texttt{type} \quad \Gamma, x : \tau \vdash e \Rightarrow \tau'}{\Gamma \vdash \lambda x : \tau.\ e \Rightarrow \tau \to \tau'}$$

3J
$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_1 \quad \tau \Downarrow \tau \to \tau' \quad \Gamma \vdash e_2 \Leftarrow \tau}{\Gamma \vdash e_1\ e_2 \Rightarrow \tau'}$$

3M
$$\frac{\Gamma \vdash e \Rightarrow \tau \quad \tau \Downarrow \forall \alpha : k.\ \tau' \quad \Gamma \vdash c \Leftarrow k}{\Gamma \vdash e\ [c] \Rightarrow [c/\alpha]\tau'}$$

3L
$$\frac{\Gamma, \alpha : k \vdash e \Rightarrow \tau}{\Gamma \vdash \Lambda \alpha : t.\ e \Rightarrow \forall \alpha : t.\ \tau}$$

3M
$$\frac{\Gamma \vdash e \Rightarrow \tau' \quad \Gamma \vdash \tau \iff \tau' : \texttt{type}}{\Gamma \vdash e \Leftarrow \tau}$$