Type Checking F_ω

Formal Methods, SSE of USTC

Spring 2015

In this assignment, you'll implement a type check for F_ω, a formal system with polymorphic types and type operators. Generally speaking, type checking F_ω is not that harder than type checking λ_ω, so we emphasize only the key difference between these two systems. And the difference parts are marked with yellow color.

1 The Syntax for F_ω

The syntax for F_ω is presented in Figure 1, with two new syntactic forms for terms: type abstraction and type application. And there is a new constructor for polymorphic types.

$$
\begin{array}{llll}
\textit{Terms} & t & \to & \texttt{true} \mid \texttt{false} \mid \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t \mid x \\
 & & \mid & \lambda x : c.t \mid t\ t \mid \lambda\alpha :: K.t \mid t\ [c] \\
\textit{Constructors} & c & \to & \texttt{Bool} \mid \alpha \mid c \to c \mid \Lambda\alpha :: K.c \mid c\ c \mid \forall\alpha :: K.c \\
\textit{Kinds} & K & \to & \star \mid K \Rightarrow K
\end{array}
$$

Figure 1: Syntax for F_ω

2 The Declarative Static Semantics for F_ω

The static semantics for F_ω consists of three components: the typing rule for terms, the kinding rules for constructors and the equivalence rules for types.

2.1 The Typing Rules

In order to present the typing rules, we first present the definition of the typing environment Γ and the kinding environment Δ, in Figure 2.

The typing rules make use of the following judgmental form

$$\Gamma; \Delta \vdash t : c,$$

Typing environment	Γ	$\rightarrow$	$\cdot$ |	$x : c, \Gamma$
Kinding environment	Δ	$\rightarrow$	$\cdot$ |	$\alpha :: K, \Delta$

Figure 2: Typing and kinding environments

and the rules are given in Figure 3. Only the T-EQ rule deserves further explanation. Essentially, this rules specifies that one can interchange a constructor c_2 when another constructor c_1 is inferable, as long as these two constructors are equivalent $c_1 \equiv c_2$, the equivalence relation $\equiv$ will be discussed shortly.

So, these typing rules are not syntax-directed and thus can not be used to direct the type checking.

2.2 The Kinding Rules

The kinding rule specifies the conditions under which a constructor is legal. These rules take the following judgmental form:

$$\Delta \vdash c :: K$$

and consists of the rules in Figure 4.

It's nice to see that the set of kinding rules are syntax-directed.

2.3 The Definitional Equivalence Rules

The equivalence relation $\equiv$ are defined on any two constructors using this judgment form:

$$\vdash c_1 \equiv c_2$$

and the rules are given in Figure 5.

It's also worth remarking that these definitional equivalence relation is not syntax-directed. For instance, when one need to compare two constructors c_1 and c_2, a feasible way is to use the T-BETA rule to try to reduce any one constructor, but another way is to use the E-TRANS rule, which involves guess a third constructor c_3. For this reason, in the next, we would develop a theory of algorithmic equivalence checking.

3 The Algorithmic Static Semantics for $\mathbf{F}_\omega$

The key step in designing an algorithmic static semantics is to make the typing rules and definitional equivalence rules syntax-directed.

The key idea to make the typing rules syntax-directed is to eliminate the T-EQ rule and move the constructor equivalence comparision to the necessary points in other typing rules. In this sense, we are providing equivalence coercion in typing rules directly. A close look at the typing rules from Figure 3 reveals that both the T-IF rule and the T-APP rule need this coercion: for the former,

$$\boxed{\Gamma; \Delta \vdash t : c}$$

$$\frac{}{\Gamma; \Delta \vdash \texttt{true} : \texttt{Bool}} \quad \text{(T-True)}$$

$$\frac{}{\Gamma; \Delta \vdash \texttt{false} : \texttt{Bool}} \quad \text{(T-False)}$$

$$\frac{\Gamma; \Delta \vdash t_1 : \texttt{Bool} \qquad \Gamma; \Delta \vdash t_2 : c \qquad \Gamma; \Delta \vdash t_3 : c}{\Gamma; \Delta \vdash \texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : c} \quad \text{(T-If)}$$

$$\frac{x : c \in \Gamma}{\Gamma; \Delta \vdash x : c} \quad \text{(T-Var)}$$

$$\frac{\Delta \vdash c :: \star \qquad \Gamma, x : c; \Delta \vdash t : c'}{\Gamma; \Delta \vdash \lambda x : c.t : c \to c'} \quad \text{(T-Abs)}$$

$$\frac{\Gamma; \Delta \vdash t_1 : c_1 \to c_2 \qquad \Gamma; \Delta \vdash t_2 : c_1}{\Gamma; \Delta \vdash t_1\ t_2 : c_2} \quad \text{(T-App)}$$

$$\frac{\Gamma; \Delta, \alpha :: K \vdash t : c}{\Gamma; \Delta \vdash \lambda\alpha :: K.t : \forall\alpha :: K.c} \quad \text{(T-TyAbs)}$$

$$\frac{\Gamma; \Delta \vdash t : \forall\alpha :: K.c' \qquad \Delta \vdash c :: K}{\Gamma; \Delta \vdash t\ [c] : [\alpha \mapsto c]c'} \quad \text{(T-TyApp)}$$

$$\frac{\Gamma; \Delta \vdash t : c_1 \qquad \vdash c_1 \equiv c_2 \qquad \Delta \vdash c_2 :: \star}{\Gamma; \Delta \vdash t : c_2} \quad \text{(T-Eq)}$$

Figure 3: Typing rules for F_ω

one need to check that the type of t_1 is really the constructor `Bool`, and that the type of t_2 and t_3 are really equivalent; and for the latter, one need to check that the term t_1 is really of an arrow type $c_1 \to c_2$ and that t_2's type is really equivalent to c_1.

$\boxed{\Delta \vdash c :: K}$

$$\frac{}{\Delta \vdash \texttt{Bool} :: \star} \quad \text{(K-TyBool)}$$

$$\frac{\Delta \vdash c_1 :: \star \qquad \Delta \vdash c_2 :: \star}{\Delta \vdash c_1 \to c_2 :: \star} \quad \text{(K-TyArrow)}$$

$$\frac{\Delta, \alpha :: K \vdash c :: \star}{\Delta \vdash \forall \alpha :: K.c :: \star} \quad \text{(K-TyForall)}$$

$$\frac{\alpha :: K \in \Delta}{\Delta \vdash \alpha :: K} \quad \text{(K-TyVar)}$$

$$\frac{\Delta, \alpha :: K_1 \vdash c :: K_2}{\Delta \vdash \Lambda \alpha :: K.c :: K_1 \Rightarrow K_2} \quad \text{(K-TyAbs)}$$

$$\frac{\Delta \vdash c_1 :: K_1 \Rightarrow K_2 \qquad \Delta \vdash c_2 :: K_1}{\Delta \vdash c_1\ c_2 :: K_2} \quad \text{(K-TyApp)}$$

Figure 4: Kinding rules for F_ω

With these in mind, we present the algorithmic typing rule via this judgmental form:

$$\Gamma; \Delta \rhd t : c$$

and the typing rules in F

We make use of two new judgmental forms:

$$\Delta \rhd c_1 \Downarrow c_2$$

and

$$\Delta \rhd c_1 \Leftrightarrow c_2 :: K$$

the former one specifies that the constructor c can reduce to another constructor c', and the latter one specifies that the two constructors c and c' are equivalent algorithmically at the kind K.

The rules for the former judgmental form is given in Figure 7. The key idea is that the β- reduction rule is applied repeatedly, until there is no constructor application exists, unless the application is to a constructor variable α. Another

$\boxed{\vdash c_1 \equiv c_2}$

$$\frac{}{\vdash c \equiv c} \quad \text{(E-Refl)}$$

$$\frac{\vdash c_1 \equiv c_2}{\vdash c_2 \equiv c_1} \quad \text{(E-Symm)}$$

$$\frac{\vdash c_1 \equiv c_2 \qquad c_2 \equiv c_3}{\vdash c_1 \equiv c_3} \quad \text{(E-Trans)}$$

$$\frac{\vdash c_1 \equiv c_3 \qquad c_2 \equiv c_4}{\vdash c_1 \to c_2 \equiv c_3 \to c_4} \quad \text{(E-Arrow)}$$

$$\frac{\vdash c_1 \equiv c_2}{\vdash \forall\alpha :: K.c_1 \equiv \forall\alpha :: K.c_2} \quad \text{(E-Forall)}$$

$$\frac{\vdash c_1 \equiv c_2}{\vdash \Lambda\alpha :: K.c_1 \equiv \Lambda\alpha :: K.c_2} \quad \text{(E-TyAbs)}$$

$$\frac{\vdash c_1 \equiv c_3 \qquad c_2 \equiv c_4}{\vdash c_1\ c_2 \equiv c_3\ c_4} \quad \text{(E-TyApp)}$$

$$\frac{}{\vdash (\Lambda\alpha :: K.c)c' \equiv [\alpha \mapsto c']c} \quad \text{(E-Beta)}$$

Figure 5: Definitional Equivalence Rules for F_ω

subtle point here is that both the constructors c and c' are of kind $\star$, and this kind is implicit in the reduction rule.

The algorithmic equivalence checking rules are given in Figure 8.

Essentially, these two rules will first push down constructors to a normal form of kind $\star$ (if they are not, we first perform η-reduction. And then we normalize these normal forms by the E-Star rule and compare c'_1 and C'_2 structurally.

This gives us the next judgmental form:

$$\Delta \rhd c_1 \leftrightarrow c_2$$

which will compare two constructors c_1 and c_2 for structural equivalence. The rules for this judgmental form are given in Figure 9.

$$\boxed{\Gamma; \Delta \rhd t : c}$$

$$\frac{}{\Gamma; \Delta \rhd \texttt{true} : \texttt{Bool}} \quad \text{(T-True)}$$

$$\frac{}{\Gamma; \Delta \rhd \texttt{false} : \texttt{Bool}} \quad \text{(T-False)}$$

$$\frac{\Gamma; \Delta \vdash t_1 : c_1 \qquad \Gamma \rhd c_1 \Downarrow \texttt{Bool} \qquad \Gamma; \Delta \vdash t_2 : c_2 \qquad \Gamma; \Delta \vdash t_3 : c_3 \qquad \Delta \rhd c_2 \Leftrightarrow c_3 :: \star}{\Gamma; \Delta \vdash \texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : c_2} \quad \text{(T-If)}$$

$$\frac{x : c \in \Gamma}{\Gamma; \Delta \rhd x : c} \quad \text{(T-Var)}$$

$$\frac{\Delta \rhd c :: \star \qquad \Gamma, x : c; \Delta \rhd t : c'}{\Gamma; \Delta \rhd \lambda x : c.t : c \to c'} \quad \text{(T-Abs)}$$

$$\frac{\Gamma; \Delta \vdash t_1 : c_1 \qquad \Delta \rhd c_1 \Downarrow c_2 \to c_3 \qquad \Gamma; \Delta \rhd t_2 : c_4 \qquad \Delta \rhd c_2 \Leftrightarrow c_4 :: \star}{\Gamma; \Delta \vdash t_1\ t_2 : c_3} \quad \text{(T-App)}$$

$$\frac{\Gamma; \Delta, \alpha :: K \rhd t : c}{\Gamma; \Delta \rhd \lambda \alpha :: K.t : \forall \alpha :: K.c} \quad \text{(T-TyAbs)}$$

$$\frac{\Gamma; \Delta \vdash t : c \qquad \Delta \rhd c \Downarrow \forall \alpha :: K.c' \qquad \Delta \rhd c'' :: K}{\Gamma; \Delta \vdash t\ [c''] : [\alpha \mapsto c'']c'} \quad \text{(T-TyApp)}$$

Figure 6: Algorithmic Typing rules for F_ω

4 The Implementation

Let's summarize, in Figure 10, all judgmental forms to type checking F_ω. Especially, we list the input, output and an interpretation of all judgmental forms. Note that in the third and fifth judgments, the kind is always $\star$ and thus are implicit.

$$\boxed{\Delta \rhd c_1 \Downarrow c_2}$$

$$\frac{}{\Delta \rhd \texttt{Bool} \Downarrow \texttt{Bool}} \quad \text{(R-Bool)}$$

$$\frac{}{\Delta \rhd c_1 \to c_2 \Downarrow c_1 \to c_2} \quad \text{(R-Arrow)}$$

$$\frac{}{\Delta \rhd \forall\alpha :: K.c \Downarrow \forall\alpha :: K.c} \quad \text{(R-Forall)}$$

$$\frac{}{\Delta \vdash \alpha \Downarrow \alpha} \quad \text{(R-TyVar)}$$

$$\frac{}{\Delta \rhd \Lambda\alpha :: K.c \Downarrow \Lambda\alpha :: K.c} \quad \text{(R-TyAbs)}$$

$$\frac{\Delta \rhd c_1 \Downarrow (\Lambda\alpha :: K.c) \qquad [\alpha \mapsto c_2]c \Downarrow c'}{\Delta \rhd c_1\ c_2 \Downarrow c'} \quad \text{(R-App1)}$$

$$\frac{\Delta \rhd c_1 \Downarrow \alpha}{\Delta \rhd c_1\ c_2 \Downarrow \alpha\ c_2} \quad \text{(R-App2)}$$

Figure 7: Reduction rules for Constructors

$$\boxed{\Delta \rhd c_1 \Leftrightarrow c_2 :: K}$$

$$\frac{\Delta \rhd c_1 \Downarrow c_1' \qquad \Delta \rhd c_2 \Downarrow c_2' \qquad \Delta \rhd c_1' \leftrightarrow c_2'}{\Delta \rhd c_1 \Leftrightarrow c_2 :: \star} \quad \text{(E-KStar)}$$

$$\frac{\Delta, \alpha :: K_1 \rhd c_1\ \alpha \Leftrightarrow c_2\ \alpha :: K_2}{\Delta \rhd c_1 \Leftrightarrow c_2 :: K_1 \Rightarrow K_2} \quad \text{(E-KArrow)}$$

Figure 8: Algorithmic Equivalence Rules for Constructors

Finally, let remark that the syntactic forms in Figure 9 are of special interest, they have been evaluated to normal forms of such a shape: all head constructors have bee exposed, but not the underlying constructors. For instance, take a look at the S-Arrow rule, the underlying constructor c_i for $1 \leq i \leq 4$ are not normal

$\boxed{\Delta \triangleright c_1 \leftrightarrow c_2}$

$$\frac{}{\Delta \triangleright \texttt{Bool} \leftrightarrow \texttt{Bool}} \quad \text{(S-Bool)}$$

$$\frac{\Delta \triangleright c_1 \Leftrightarrow c_3 :: \star \qquad \Delta \triangleright c_2 \Leftrightarrow c_4 :: \star}{\Delta \triangleright c_1 \rightarrow c_2 \leftrightarrow c_3 \rightarrow c_4} \quad \text{(S-Arrow)}$$

$$\frac{\Delta \triangleright c_1 \Leftrightarrow c_2 :: \star}{\Delta \triangleright \forall \alpha :: K.c_1 \leftrightarrow \forall \alpha :: K.c_2} \quad \text{(S-Forall)}$$

$$\frac{}{\Delta \triangleright \alpha \leftrightarrow \alpha} \quad \text{(S-TyVar)}$$

$$\frac{\Delta \triangleright \alpha :: K_1 \Rightarrow \star \qquad \Delta \triangleright c_1 \Leftrightarrow c_2 :: K_1}{\Delta \triangleright \alpha\ c_1 \leftrightarrow \alpha\ c_2} \quad \text{(S-TyApp)}$$

Figure 9: Structural Equivalence Rules for Constructors

The judgment	Input	Output	Interpretation
$\Gamma; \Delta \triangleright t : c$	Γ, Δ, t	c	Type checking a term t
$\Delta \triangleright c :: K$	Δ, c	K	Kind checking a constructor c
$\Delta \triangleright c_1 \Downarrow c_2$	Δ, c_1	c_2	β-reduce a constructor c_1
$\Delta \triangleright c_1 \Leftrightarrow c_2 :: K$	Δ, c_1, c_2, K	boolean	algorithmic equivalence
$\Delta \triangleright c_1 \leftrightarrow c_2$	Δ, c_1, c_2	boolean	structural equivalence

Figure 10: All Judgmental Forms

forms can thus can be reduced further. Such kind of normal forms are called *weak head normal forms* in the literatures.