Tethys

A Toy Functional Programming Language
with a System Fω-based Core Calculus
https://github.com/ThePuzzlemaker/tethys

The Puzzlemaker

August 18, 2021

Contents

1 Introduction

This “paper” (which is really just a well-typeset, but somewhat informal write-up) introduces Tethys, a toy functional programming language based on a System Fω-based core calculus. Hence the title.

There are two parts of Tethys: the surface language, and the core calculus. The core calculus is the intermediate representation of Tethys which is used for type checking and inference, and for interpretation. The surface language is the higher-level interface that is eventually desugared by the compiler/interpreter to the core calculus.

The reference implementation in Rust will not use any particular “tricks” in terms of interpretation, instead just using a simple tree-walk interpreter or similar.

This language was created in order to conduct informal research (i.e., not actually discovering anything interesting, probably) on type systems, especially bidirectional typechecking and polymorphism. Tethys is named as such as it is the name of the co-orbital moon to Calypso; as my work on this language is “co-orbital“, so to speak, to my work on Calypso.[1]

2 The Surface Language

This section has not been started yet.

3 The Core Calculus

This section is, unsurprisingly, work-in-progress.

3.1 Abstract Syntax

$$
\begin{array}{lll}
\kappa ::= & & \text{kinds:} \\
 & * & \text{concrete types} \\
 & \kappa \to \kappa & \text{type constructors}
\end{array}
$$

$$
\begin{array}{lll}
\sigma, \tau ::= & & \text{monotypes:} \\
 & \alpha & \text{type variables} \\
 & \texttt{bool} & \text{booleans} \\
 & \texttt{int} & \text{64-bit signed integer} \\
 & \{\overline{\tau_i}\} & n\text{-ary products} \\
 & \langle\overline{\tau_i}\rangle & n\text{-ary sums} \\
 & \sigma \to \tau & \text{arrows} \\
 & \mu\alpha.\tau & \text{recursive types}
\end{array}
$$

[1] More information on Calypso (the language, of course) is available at https://calypso-lang.github.io

$A, B, C ::=$		types:
	σ, τ	monotypes
	$\forall(\alpha :: \kappa).A$	universal quantification
	$\lambda(\alpha :: \kappa).A$	type-level lambdas (type constructor)
	$A\ B$	type-level application

$t ::=$		terms:
	x	variables
	c	constants
	$e : A$	type annotation
	$\{\overline{e_i}\}$	introduce product
	$e.i$	product elimination (projection)
	$\langle i = e \rangle$ **as** A	variant introduction (injection)
	case e **of** e	variant elimination
	$\lambda x.e$	lambda abstraction
	$e\ e$	lambda application
	$e[\tau]$	type application
	$\texttt{unfold}_{\mu\alpha.\tau} : \mu\alpha.\tau \to \tau[\mu\alpha.\tau/\alpha]$	isorecursive folding
	$\texttt{fold}_{\mu\alpha.\tau} : \tau[\mu\alpha.\tau/\alpha] \to \mu\alpha.\tau$	isorecursive unfolding

$c ::=$		constants and builtins:
	$\texttt{fix} : \forall\alpha.(\alpha \to \alpha) \to \alpha$	fixpoint combinator
	$\texttt{true}, \texttt{false}$	boolean literals
	$\ldots, -2, -1, 0, 1, 2, \ldots$	integer literals
	$+, -, *, /, \texttt{mod} : \texttt{int} \to \texttt{int} \to \texttt{int}$	arithmetic operators
	$\texttt{and}, \texttt{or} : \texttt{bool} \to \texttt{bool} \to \texttt{bool}$	binary boolean operators
	$\texttt{not} : \texttt{bool} \to \texttt{bool}$	negation
	$\texttt{cond} : \forall\alpha.\texttt{bool} \to \alpha \to \alpha$	conditional

$\Gamma ::=$		contexts:
	$\emptyset$	empty context
	$\Gamma, x : A$	variable binding

$\Gamma, \alpha :: \kappa$	type variable binding
$\Gamma, \hat{\alpha}$	existential type variable
$\Gamma, \hat{\alpha} = \tau$	solved existential type variable

3.2 Kinding Rules

$$\frac{}{\texttt{bool} :: *}\ (\textsc{K-Bool}) \qquad \frac{}{\texttt{int} :: *}\ (\textsc{K-Int}) \qquad \frac{\Gamma \vdash \tau_i :: * \text{ (for all } i)}{\Gamma \vdash \{\overline{\tau_i}\} :: *}\ (\textsc{K-Sum})$$

$$\frac{\Gamma \vdash \tau_i :: * \text{ (for all } i)}{\Gamma \vdash \langle\overline{\tau_i}\rangle :: *}\ (\textsc{K-Sum}) \qquad \frac{\Gamma \vdash \sigma :: * \qquad \Gamma \vdash \tau :: *}{\Gamma \vdash \sigma \to \tau :: *}\ (\textsc{K-Arr})$$

$$\frac{\Gamma, \alpha :: \kappa \vdash \tau :: \kappa}{\Gamma \vdash \mu\alpha.\tau :: \kappa}\ (\textsc{K-Fix}) \qquad \frac{\alpha :: \kappa \in \Gamma}{\Gamma \vdash \alpha :: \kappa}\ (\textsc{K-Var}) \qquad \frac{\Gamma, \alpha :: \kappa \vdash A :: *}{\Gamma \vdash \forall(\alpha :: \kappa).A :: *}\ (\textsc{K-All})$$

$$\frac{\Gamma, \alpha :: \kappa \vdash A :: \kappa'}{\Gamma \vdash \lambda(\alpha :: \kappa).A :: \kappa \to \kappa'}\ (\textsc{K-Lam}) \qquad \frac{\Gamma \vdash \tau_1 :: \kappa \to \kappa' \qquad \Gamma \vdash \tau_2 :: \kappa'}{\Gamma \vdash \tau_1\ \tau_2 :: \kappa'}\ (\textsc{K-App})$$

3.3 Well-Formedness of Types, Kinds, and Contexts

$\boxed{\Gamma \vdash A}$ Under context Γ, type A is well-formed

$$\frac{}{\Gamma \vdash \texttt{bool}}\ (\textsc{WF-Bool}) \qquad \frac{}{\Gamma \vdash \texttt{int}}\ (\textsc{WF-Int}) \qquad \frac{\Gamma \vdash \tau_i \text{ (for all } i)}{\Gamma \vdash \{\overline{\tau_i}\}}\ (\textsc{WF-Prod})$$

$$\frac{\Gamma \vdash \tau_i \text{ (for all } i)}{\Gamma \vdash \langle\overline{\tau_i}\rangle}\ (\textsc{WF-Sum}) \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \to B}\ (\textsc{WF-Arr})$$

$$\frac{\Gamma, \alpha \vdash \tau}{\Gamma \vdash \mu\alpha.\tau}\ (\textsc{WF-Fix}) \qquad \frac{}{\Gamma, \alpha \vdash \alpha}\ (\textsc{WF-Uvar}) \qquad \frac{}{\Gamma, \hat{\alpha} \vdash \hat{\alpha}}\ (\textsc{WF-Evar})$$

$$\frac{\Gamma \vdash \tau}{\Gamma, \hat{\alpha} = \tau \vdash \hat{\alpha}}\ (\textsc{WF-SolvedEvar}) \qquad \frac{\Gamma, \alpha :: \kappa \vdash A}{\Gamma \vdash \forall(\alpha :: \kappa).A}\ (\textsc{WF-Forall})$$

$$\frac{\Gamma, \alpha :: \kappa \vdash A}{\Gamma \vdash \lambda(\alpha :: \kappa).A}\ (\textsc{WF-Lambda}) \qquad \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A\ B}\ (\textsc{WF-App})$$

$\boxed{\Gamma \vdash \kappa}$ Under context Γ, kind κ is well-formed

$$\frac{}{\Gamma \vdash *}\ (\textsc{KWF-Star}) \qquad \frac{\Gamma \vdash \kappa \qquad \Gamma \vdash \kappa'}{\Gamma \vdash \kappa \to \kappa'}\ (\textsc{KWF-Cons})$$

$\boxed{\Gamma \; ctx}$ Context Γ is well-formed

$$\frac{}{\emptyset \; ctx} \; (\textsc{Ctx-Empty}) \qquad \frac{\Gamma \; ctx \qquad x \notin \mathrm{dom}(\Gamma) \qquad \Gamma \vdash A}{\Gamma, x : A \; ctx} \; (\textsc{Ctx-Var})$$

$$\frac{\Gamma \; ctx \qquad \alpha \notin \mathrm{dom}(\Gamma) \qquad \Gamma \vdash \kappa}{\Gamma, \alpha :: \kappa \; ctx} \; (\textsc{Ctx-Uvar}) \qquad \frac{\Gamma \; ctx \qquad \alpha \notin \mathrm{dom}(\Gamma)}{\Gamma, \hat{\alpha} \; ctx} \; (\textsc{Ctx-Evar})$$

$$\frac{\Gamma \; ctx \qquad \alpha \notin \mathrm{dom}(\Gamma) \qquad \Gamma \vdash \tau}{\Gamma, \hat{\alpha} = \tau \; ctx} \; (\textsc{Ctx-SolvedEvar})$$