Three toy type systems

davidad

2018–03–31

0 Introduction

0.1 Purpose

This document represents the fulfillment of the following task: to identify and/or develop 3 extremely small (“toy”) type systems, which despite their minimal complexity demonstrate dramatically different approaches to typing: old-style static typing, static typing with generics, and dynamic typing. The purpose of this exercise is to use these type systems as examples for reasoning about how future system designs under the IPLD umbrella might, in a principled and theoretically sound way, enable interoperability between code written in languages with such dramatically different approaches to typing as Go, Rust, and JavaScript.

0.2 Overview

In this document, we present, using uniform notation, three such type systems:

1. The first, System F, also known as second-order lambda calculus, is a well-studied extension of the simply-typed λ-calculus to handle parametric polymorphism (generics), and is the sort of system that would likely be known by aliens—it is so natural that it was independently discovered by Jean-Yves Girard [Gir72] and John Reynolds [Rey74] two years apart.
2. The second, Featherweight Java (or FJ), was introduced in [IPW01] to “bear a similar relation to Java as the λ-calculus does to languages such as ML and Haskell,” presented here with minor alterations. Unlike System F, it includes class-based inheritance, but it does not include generics.[1]
3. The final type system, Featherweight JavaScript (or FJS), is a “uni-typed” system (the only type is Expr) which gives syntax to a tiny language inspired by [Zha16], with the addition of “objects” (dictionaries).

Note that at this stage, we are not concerning ourselves with mismatches in the primitive types. To make this explicit, we supply Num and Bool as the primitive types for all three type systems, with only superficial differences.

[1] Although it is extremely simplified from Java, I would still consider it unlikely to be in use by aliens.

1 Type system F (static with generics)

Simply-typed λ-calculus:

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash (\lambda x : A.e) : A \to B}\textsc{Abs} \qquad \frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}\textsc{Var}$$

$$\frac{\Gamma \vdash x : A \to B \quad \Gamma \vdash y : A}{\Gamma \vdash xy : B}\textsc{App}$$

System F:

$$\frac{\Gamma \vdash t : A \quad \alpha \notin \Gamma}{\Gamma \vdash \Lambda\alpha.t : \forall\alpha.A}\text{F-}\textsc{TAbs}$$

$$\frac{\Gamma \vdash t : \forall\alpha.A}{\Gamma \vdash tB : A[\alpha \to B]}\text{F-}\textsc{TApp}$$

As in the untyped λ-calculus, product types and sum types can be encoded into System F as appropriate polymorphic function types (using the principles of Church encoding):

$$A \times B := \forall\alpha.(A \to B \to \alpha) \to \alpha$$

$$A + B := \forall\alpha.(A \to \alpha) \to (B \to \alpha) \to \alpha$$

Even recursive types can be so encoded, for example the generic type of lists:

$$\mathsf{List}\,A := \forall\alpha.(A \to (\alpha \to \alpha)) \to (\alpha \to \alpha)$$

However, in the interest of maintaining uniformity of primitive types, we define non-Church-encoded Bool and Num, with a simplified set of operators, as follows.

Primitive types:

$$\frac{}{\Gamma \vdash \texttt{true} : \mathsf{Bool}} \qquad \frac{}{\Gamma \vdash \texttt{false} : \mathsf{Bool}}$$

$$\frac{\Gamma \vdash x : \mathsf{Bool} \quad \Gamma \vdash e_1 : A \quad \Gamma \vdash e_2 : A}{\Gamma \vdash (\texttt{if}\ x\ \texttt{then}\ e_1\ \texttt{else}\ e_2) : A}$$

$$\frac{\Gamma \vdash x : \mathsf{Num} \quad \Gamma \vdash y : \mathsf{Num}}{\Gamma \vdash x + y : \mathsf{Num}} \qquad \frac{\Gamma \vdash x : \mathsf{Num} \quad \Gamma \vdash y : \mathsf{Num}}{\Gamma \vdash x - y : \mathsf{Num}}$$

$$\frac{\Gamma \vdash x : \mathsf{Num} \quad \Gamma \vdash y : \mathsf{Num}}{\Gamma \vdash x * y : \mathsf{Num}} \qquad \frac{\Gamma \vdash x : \mathsf{Num} \quad \Gamma \vdash y : \mathsf{Num}}{\Gamma \vdash x / y : \mathsf{Num}}$$

$$\frac{n \in \mathbb{N}}{\Gamma \vdash \ulcorner n \urcorner : \mathsf{Num}} \qquad \frac{\Gamma \vdash x : \mathsf{Num}}{\Gamma \vdash x > 0 : \mathsf{Bool}}$$

2 Type system FJ (static with inheritance)

Expression typing:

$$\frac{(x:A)\in\Gamma}{\Gamma\vdash x:A}\,\textsc{Var}$$

$$\frac{\Gamma\vdash x:A \quad \mathit{fields}(A)\ni B\ \texttt{f}}{\Gamma\vdash x\,.\,\texttt{f}:B}\,\text{FJ-}\textsc{Field}$$

$$\frac{\Gamma\vdash x:A \quad \overline{\Gamma\vdash e:B} \quad \overline{B<:C} \quad \mathit{mtype}(\texttt{m},A)=\overline{C}\to D}{\Gamma\vdash x\,.\,\texttt{m}(\overline{e}):D}\,\text{FJ-}\textsc{Invk}$$

$$\frac{\mathit{fields}(A)=\overline{C\ \texttt{f}} \quad \overline{\Gamma\vdash e:B} \quad \overline{B<:C}}{\Gamma\vdash \texttt{new}\,A(\overline{e}):A}\,\text{FJ-}\textsc{New}$$

$$\frac{\Gamma\vdash e:A \quad A<:B}{\Gamma\vdash (B)e:B}\,\text{FJ-UC}\textsc{ast} \qquad \frac{\Gamma\vdash e:A \quad B<:A \quad B\neq A}{\Gamma\vdash (B)e:B}\,\text{FJ-DC}\textsc{ast}$$

$$\frac{\Gamma\vdash e:A \quad A\not<:B \quad B\not<:A \quad \text{illegal cast}}{\Gamma\vdash (B)e:B}\,\text{FJ-SC}\textsc{ast}$$

Subtyping:

$$\frac{}{C<:C} \qquad \frac{\texttt{class}\ C\ \texttt{extends}\ D \quad D<:E}{C<:E}$$

Method checking:

$$\frac{\overline{x:T},\texttt{this}:C\vdash e:A \quad A<:B \quad \texttt{class}\ C\ \texttt{extends}\ D \quad \Big(\mathit{mtype}(\texttt{m},D)=\overline{E}\to F\Big)\Rightarrow\overline{T=E}, B=F}{B\ \ \texttt{m}(\overline{T\ x})\texttt{\{ return }e\texttt{; \}}\quad \text{OK in } C}\,\text{FJ-}\textsc{Method}$$

Class checking:

$$\frac{K=C\,(\overline{D\texttt{g}},\ \overline{C\texttt{f}})\texttt{\{super(}\overline{\texttt{g}}\texttt{);}\ \overline{\texttt{this.f=f;}}\texttt{\}} \quad \mathit{fields}(D)=\overline{D\texttt{g}} \quad \overline{M\ \text{OK in}\ C}}{\texttt{class}\ C\ \texttt{extends}\ D\ \overline{C\texttt{f}};\ K\ \overline{M}\quad \text{OK}}\,\text{FJ-}\textsc{Class}$$

Primitive types:

$$\frac{}{\Gamma\vdash\texttt{true}:\mathsf{Bool}} \qquad \frac{}{\Gamma\vdash\texttt{false}:\mathsf{Bool}}$$

$$\frac{\Gamma\vdash x:\mathsf{Bool} \quad \Gamma\vdash e_1:A \quad \Gamma\vdash e_2:A}{\Gamma\vdash(x\ ?\ e_1\ :\ e_2):A}$$

$$\frac{\Gamma\vdash x:\mathsf{Num} \quad \Gamma\vdash y:\mathsf{Num}}{\Gamma\vdash x+y:\mathsf{Num}} \qquad \frac{\Gamma\vdash x:\mathsf{Num} \quad \Gamma\vdash y:\mathsf{Num}}{\Gamma\vdash x-y:\mathsf{Num}}$$

$$\frac{\Gamma\vdash x:\mathsf{Num} \quad \Gamma\vdash y:\mathsf{Num}}{\Gamma\vdash x*y:\mathsf{Num}} \qquad \frac{\Gamma\vdash x:\mathsf{Num} \quad \Gamma\vdash y:\mathsf{Num}}{\Gamma\vdash x\,/\,y:\mathsf{Num}}$$

$$\frac{n\in\mathbb{N}}{\Gamma\vdash\ulcorner n\urcorner:\mathsf{Num}} \qquad \frac{\Gamma\vdash x:\mathsf{Num}}{\Gamma\vdash x>\mathtt{0}:\mathsf{Bool}}$$

3 Type system FJS (dynamic)

This “type system” actually provides *no* static type-checking at all; however, it is less dynamic than JavaScript in one respect: it does statically check whether variables are in scope! But it does not check for, e.g., the existence of fields being retrieved or updated, let alone rejecting expressions like `false + 5`. (Such expressions would, of course, result in run-time errors; like the other systems presented here, however, run-time semantics is not yet formally specified.)

Functions:

$$\frac{\Gamma, x : \mathsf{Expr} \vdash e : \mathsf{Expr}}{\Gamma \vdash \texttt{function}\,(\overline{x})\ \{e\} : \mathsf{Expr}}\ \text{FJS-}\textsc{Abs} \qquad \frac{(x : \mathsf{Expr}) \in \Gamma}{\Gamma \vdash x : \mathsf{Expr}}\ \text{FJS-}\textsc{Var}$$

$$\frac{\Gamma \vdash x : \mathsf{Expr} \quad \overline{\Gamma \vdash y : \mathsf{Expr}}}{\Gamma \vdash x\,(\overline{y}) : \mathsf{Expr}}\ \text{FJS-}\textsc{App}$$

Objects:

$$\frac{}{\Gamma \vdash \texttt{\{\}} : \mathsf{Expr}}\ \text{FJS-}\textsc{EmptyObj}$$

$$\frac{\Gamma \vdash x : \mathsf{Expr}}{\Gamma \vdash x\,.\texttt{f} : \mathsf{Expr}}\ \text{FJS-}\textsc{Field} \qquad \frac{\Gamma \vdash x : \mathsf{Expr} \quad \Gamma \vdash e : \mathsf{Expr}}{\Gamma \vdash x\,[\texttt{f} := e] : \mathsf{Expr}}\ \text{FJS-}\textsc{FieldUpdate}$$

Primitive types:

$$\frac{}{\Gamma \vdash \texttt{true} : \mathsf{Expr}} \qquad \frac{}{\Gamma \vdash \texttt{false} : \mathsf{Expr}}$$

$$\frac{\Gamma \vdash x : \mathsf{Expr} \quad \Gamma \vdash e_1 : \mathsf{Expr} \quad \Gamma \vdash e_2 : \mathsf{Expr}}{\Gamma \vdash (x\ ?\ e_1\ :\ e_2) : \mathsf{Expr}}$$

$$\frac{\Gamma \vdash x : \mathsf{Expr} \quad \Gamma \vdash y : \mathsf{Expr}}{\Gamma \vdash x + y : \mathsf{Expr}} \qquad \frac{\Gamma \vdash x : \mathsf{Expr} \quad \Gamma \vdash y : \mathsf{Expr}}{\Gamma \vdash x - y : \mathsf{Expr}}$$

$$\frac{\Gamma \vdash x : \mathsf{Expr} \quad \Gamma \vdash y : \mathsf{Expr}}{\Gamma \vdash x * y : \mathsf{Expr}} \qquad \frac{\Gamma \vdash x : \mathsf{Expr} \quad \Gamma \vdash y : \mathsf{Expr}}{\Gamma \vdash x \,/\, y : \mathsf{Expr}}$$

$$\frac{n \in \mathbb{N}}{\Gamma \vdash \ulcorner n \urcorner : \mathsf{Expr}} \qquad \frac{\Gamma \vdash x : \mathsf{Expr}}{\Gamma \vdash x > 0 : \mathsf{Expr}}$$

References

[Gir72] Jean-Yves Girard. “Interprétation fonctionelle et élimination des coupures danse l’arithmétique d’ordre supérieur”. PhD thesis. Université Paris 7, 1972.

[Rey74] John C. Reynolds. “Towards a theory of type structure”. In: *Programming Symposium. Colloque sur la Programmation*. (Paris, Apr. 9–11, 1974). Vol. 19. LNCS. Springer-Verlag, 1974, pp. 408–423.

[IPW01] Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. “Featherweight Java: a minimal core calculus for Java and GJ”. In: *ACM Transactions on Programming Languages and Systems* 23.3 (May 2001), pp. 396–450. DOI: 10.1145/503502.503505.

[Zha16] Keyan Zhang. *The Super Tiny Interpreter*. 2016. URL: https://github.com/keyanzhang/the-super-tiny-interpreter.