Potatoes and Parametricity

SSSH, IT'S A SECRET

ACM Reference Format:
sssh, it's a secret. 2019. Potatoes and Parametricity. 1, 1 (December 2019), 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

When ignorance is bliss, 'tis folly to be wise. This paper is about the formal metatheory of dependent type systems, presented *bidirectionally*, which introduces a pathetic programming language, expressly for the purpose of maintaining useful cluelessness by brute force. MetaSpud is a kind of machine code for components of type systems, crafted carefully to ensure that key metatheoretical properties come as *theorems for free*. If you can't do it right, then you can't do it at all.

MetaSpud is implemented in Agda, exploiting the fact that Agda's typechecker executes programs, the MetaSpud interpreter in this case, to determine the meanings of types. MetaSpud's parametricity theorem, proven in Agda, is the workhorse of a library of metatheoretical tools.

We work with a class of type theories, all of which share a common pair of syntaxes:

$$\begin{array}{lll} \textbf{constructions}\ R,S,T,r,s,t & ::= & \mathrm{a} \\ & \mid & s,t \\ & \mid & x.t \\ & \mid & \underline{e} \end{array} \qquad \begin{array}{lll} \textbf{computations}\ e,f & ::= & x \\ & \mid & e\ s \\ & \mid & t : T \end{array}$$

The **constructions** are made of the raw lumpen first-order inductive stuff of values and types—the potatoes of this piece—atoms a, pairs s,t, abstractions $x.t$, but they also embed the **computations** $\underline{e}$ which have not yet reached a value, either because they are variables x as yet unknown, or because they are eliminations $e\ s$ as yet undone. The computations, in turn, embed the constructions, but only in the form of *radicals* $t : T$, which are values ready to compute, empowered by a type annotation.

Atoms are symbols. They can be any set with a decidable equality. There must be one particular atom, $\star$, whose job is to be the type of types. Pairs nest to the right and bind more tightly than abstractions. Radicals are the loosest of all, so that

$$x.t : \Pi, S, x.T \qquad \text{means} \qquad (x.t) : (\Pi, (S, (x.T)))$$

In real life, abstractions are implemented using de Bruijn variables in a well scoped representation, but let us be informal in print and write names.

Contexts Γ assign types to variables $x : S$, but is our careful habit not to talk about whole contexts in typing rules when we can act on them by local extension. We trade in judgments J of the forms

type checking $T \ni t$ for constructions
type synthesis $e \in S$ for computations
single-step reduction $t \rightsquigarrow t'$ and $e \rightsquigarrow e'$ for both

Author's address: sssh, it's a secret.

XXXX-XXXX/2019/12-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

context lookup $\dashv x : S$
local extension $x : S \vdash J$

The following rules for type checking and synthesis are fixed for all systems.

$$\textbf{pre-reduce}\ \frac{T \rightsquigarrow T' \quad T' \ni t}{T \ni t} \qquad \textbf{post-reduce}\ \frac{e \in S \quad S \rightsquigarrow S'}{e \in S'}$$

$$\textbf{embed}\ \frac{e \in S \quad S = T}{T \ni \underline{e}} \qquad \textbf{variable}\ \frac{\dashv x : S}{x \in S}$$

$$\textbf{radical}\ \frac{\star \ni T \quad T \ni t}{t : T \in T}$$

Likewise, single step reduction includes the υ-steps which strip the type annotation from a value once its computation is done, and is closed under all subterm contexts.

$$\textbf{upsilon} \qquad \frac{}{\underline{t : T} \rightsquigarrow t}$$

$$\frac{s \rightsquigarrow s'}{s, t \rightsquigarrow s, t'} \qquad \frac{t \rightsquigarrow t'}{s, t \rightsquigarrow s, t'} \qquad \frac{t \rightsquigarrow t'}{x.t \rightsquigarrow x.t'}$$

$$\frac{e \rightsquigarrow e'}{e\ s \rightsquigarrow e'\ s} \qquad \frac{s \rightsquigarrow s'}{e\ s \rightsquigarrow e'\ s} \qquad \frac{t \rightsquigarrow t'}{t : T \rightsquigarrow t' : T} \qquad \frac{T \rightsquigarrow T'}{t : T \rightsquigarrow t' : T}$$

Beyond this basic setup, however, we make no fixed choices about which constructions are classified by which other constructions, or about how eliminated radicals should compute by β-rules of the form

$$(t : T)\ s \rightsquigarrow r : R$$

You are free to craft these rules however you like, as long as what you like are potatoes. We equip you with MetaSpud, a programming language for implementing typing and reduction rules, with limited power to inspect the syntax—we inspect only the outer atoms-pairs-abstractions structure of terms. It provides enough power for sensible rules like these, for functions:

$$\frac{\star \ni S \quad x : S \vdash \star \ni T}{\star \ni \Pi, S, x.T} \qquad \frac{x : S \vdash T \ni t}{\Pi, S, x.T \ni x.t} \qquad \frac{e \in \Pi, S, x.T \quad S \ni s}{e\ s \in T[s : S/x]}$$

$$\frac{}{(x.t : \Pi, S, x.T)\ s \rightsquigarrow (t : T)[s : S/x]}$$