Type theory in type theory using single substitutions

Ambrus Kaposi and Szumi Xie

Eötvös Loránd University, Budapest, Hungary, `{akaposi|szumi}@inf.elte.hu`

In this abstract we define type theory in a minimalistic way: we aim for an intrinsic, that is, well-typed and quotiented definition of the syntax, that is as small as possible in terms of number of operations and equations for the substitution calculus. We only introduce operations that are forced upon us. We aim for a language with Π types, and it turns out that we have to add all the ingredients of the substitution calculus while introducing the usual rules for Π types (abstraction, application, β and η). When adding more type formers, we don't need to extend the substitution calculus anymore.

We need variables in our language, so we introduce sorts of contexts, types and variables.

$$\mathsf{Con} : \mathsf{Set} \qquad\qquad \mathsf{Ty} : \mathsf{Con} \to \mathsf{Set} \qquad\qquad \mathsf{Var} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}$$

Contexts are either empty or are built from a context extended with a type.

$$\diamond : \mathsf{Con} \qquad\qquad - \triangleright - : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$$

We define variables as well-typed De Bruijn indices, but to express these we need to weaken types: e.g. the zero De Bruijn index vz has a weakened type. We introduce a new sort for substitutions Sub, an instantiation operation $-[-]$ on types, and a weakening substitution p. For now, Sub seems like an overkill because we are only using $-[\mathsf{p}]$, but it will come handy soon.

$$\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \qquad -[-] : \mathsf{Ty}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta \qquad \mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma$$

$$\mathsf{vz} : \mathsf{Var}\,(\Gamma \triangleright A)\,(A[\mathsf{p}]) \qquad \mathsf{vs} : \mathsf{Var}\,\Gamma\,A \to \mathsf{Var}\,(\Gamma \triangleright B)\,(A[\mathsf{p}])$$

Now we introduce Π types together with an equation on how instantiation with p acts on them. This is tricky: as Π binds a new variable in its second argument, we need a new version of the weakening substitution which leaves the last variable untouched. This is why we introduce lifting of a substitution $-^+$, and now we can state a general instantiation rule for Π which works not only for p, but arbitrary substitutions (including lifted ones).

$$\Pi : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma \qquad -^+ : (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Sub}\,(\Delta \triangleright A[\gamma])\,(\Gamma \triangleright A)$$
$$\Pi[] : (\Pi\,A\,B)[\gamma] = \Pi\,(A[\gamma])\,(B[\gamma^+])$$

In addition to having variables, we need a sort of terms which includes variables and lambda abstraction.

$$\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set} \qquad \mathsf{var} : \mathsf{Var}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \qquad \mathsf{lam} : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$$

To express application, we need single substitutions as well because the argument of the function appears in the return type. In addition to p and $-^+$, $\langle - \rangle$ is the third and last operation for creating substitutions.

$$\langle - \rangle : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright A) \qquad - \cdot - : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to (a : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(B[\langle a \rangle])$$

Now we would like to express the β computation rule, but for this we also need to be able to instantiate terms (in addition to types).

$$-[-] : \mathsf{Tm}\,\Gamma\,A \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma]) \qquad \Pi\beta : \mathsf{lam}\,t \cdot a = t[\langle a \rangle]$$

Now that we have instantiation of terms, we need to revisit all operations producing terms and provide rules on how to instantiate them: first of all, we need instantiation rules for lam and $-\cdot-$. The rule $\mathsf{lam}[]$ is well-typed because of $\Pi[]$, however $\cdot[]$ is not well-typed on its own and requires a new equation $[\langle\rangle]$.

$$\mathsf{lam}[] : (\mathsf{lam}\,t)[\gamma] = \mathsf{lam}\,(t[\gamma^+]) \quad [\langle\rangle] : A[\langle a\rangle][\gamma] = A[\gamma^+][\langle a[\gamma]\rangle] \quad \cdot[] : (t\cdot a)[\gamma] = (t[\gamma])\cdot(a[\gamma])$$

Then we need instantiation rules for variables, we list these for each possible substitution: weakening of a variable increases the index by one; when instantiating with lifted substitutions and single substitutions, we have to do case distinction on the De Bruijn index whether it is zero or successor. For the latter two cases, we need type equations (named $[\mathsf{p}][^+]$ and $[\mathsf{p}][\langle\rangle]$) to typeckeck the term equations.

$$\begin{array}{lll} & \mathsf{var}\,x[\mathsf{p}] = \mathsf{var}\,(\mathsf{vs}\,x) & \\ [\mathsf{p}][^+] : A[\mathsf{p}][\gamma^+] = A[\gamma][\mathsf{p}] & \mathsf{var}\,\mathsf{vz}[\gamma^+] = \mathsf{var}\,\mathsf{vz} & \mathsf{var}\,(\mathsf{vs}\,x)[\gamma^+] = \mathsf{var}\,x[\gamma][\mathsf{p}] \\ [\mathsf{p}][\langle\rangle] : A[\mathsf{p}][\langle a\rangle] = A & \mathsf{var}\,\mathsf{vz}[\langle a\rangle] = a & \mathsf{var}\,(\mathsf{vs}\,x)[\langle a\rangle] = \mathsf{var}\,x \end{array}$$

Finally, to typecheck the $\Pi\eta$ rule, we need our last equation on types.

$$[\mathsf{p}^+][\langle\mathsf{vz}\rangle] : A[\mathsf{p}^+][\langle\mathsf{var}\,\mathsf{vz}\rangle] = A \qquad\qquad \Pi\eta : t = \mathsf{lam}\,(t[\mathsf{p}]\cdot\mathsf{var}\,\mathsf{vz})$$

This concludes all the rules for type theory with Π. We summarise as follows: there are three kinds of substitutions (single weakening, single substitution, lifted substitution), we have 5 equations describing how instantiation acts on variables, and 4 equations which describe general properties of instantiation on types. The rest of the rules are specific to our single type former Π: the only extra requirement is that each operation is equipped with an instantiation rule ($\Pi[]$, $\mathsf{lam}[]$, $\cdot[]$). Perhaps surprisingly, this is enough to define the syntax: there is no need for Con and Sub to a form a category, no need for parallel substitutions, empty substitution, parallel weakenings, or other combinations of these.

When adding new type formers, we only need the rules for the type former, and an extra instantiation (naturality) rule for each operation. For example, a Coquand-universe can be added by $\mathsf{U} : \mathsf{Ty}\,\Gamma$, $\mathsf{El} : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma$, $\mathsf{c} : \mathsf{Ty}\,\Gamma \to \mathsf{Tm}\,\Gamma\,\mathsf{U}$, $\mathsf{U}\beta : \mathsf{El}\,(\mathsf{c}\,A) = A$, $\mathsf{U}\eta : \mathsf{c}\,(\mathsf{El}\,a) = a$, and three instantiation rules (note that we need indexing to avoid inconsistency). In [4], we showed that any second-order generalised algebraic theory (SOGAT) has a single substitution presentation.

In the syntax (initial model, quotient inductive-inductive type) of the above theory, all the rules of categories with families (CwF [3]) are admissible. That is, by induction on the single substitution syntax, we can define parallel substitutions (lists of terms) which are composable and form a category; we can define instantiation by parallel substitutions for types and terms, these have the usual universal property of comprehension. The main ingredient for this construction is the notion of α-normal form: a kind of normal form which is still quotiented by $\Pi\beta$, $\Pi\eta$, but does not include explicit substitutions. If a type is in α-normal form, we know whether it is Π, U, or El of a term. If a term is in α-normal form, we know whether it is a variable, a lam, an application or a code for a type (note that any function on an α-normal has to respect $\Pi\beta$, $\Pi\eta$, $\mathsf{U}\beta$, $\mathsf{U}\eta$). We prove α-normalisation (every term has a unique α-normal form) and then use induction on α-normal forms to define parallel instantiation and prove its properties.

We formalised the single substitution calculus with an infinite hierarchy of types closed under Π and U as a quotient inductive-inductive type in Agda, proved α-normalisation for it, and derived all the rules for the parallel substitution calculus (CwF equipped with Π types and U).

It is clear that the rules for the single substitution calculus are all derivable from the CwF-rules. The other direction is however not true: there are more models of the single substitution calculus than the parallel substitution calculus. The situation is analogous to the relationship of combinatory logic and lambda calculus: the former has more models, but the syntaxes are isomorphic [2]. Our single substitution calculus is a minimalistic version of B-systems [1] (C-systems are their parallel substitution analogue): B-systems include telescopes and more equations than our single substitution calculus, making them equivalent to C-systems.

References

[1] Benedikt Ahrens, Jacopo Emmenegger, Paige Randall North, and Egbert Rijke. B-systems and C-systems are equivalent. *The Journal of Symbolic Logic*, page 1–9, 2023. doi:10.1017/jsl.2023.41.

[2] Thorsten Altenkirch, Ambrus Kaposi, Artjoms Sinkarovs, and Tamás Végh. Combinatory logic and lambda calculus are equal, algebraically. In Marco Gaboardi and Femke van Raamsdonk, editors, *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy*, volume 260 of *LIPIcs*, pages 24:1–24:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.FSCD.2023.24, doi:10.4230/LIPICS.FSCD.2023.24.

[3] Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. *CoRR*, abs/1904.00827, 2019. URL: http://arxiv.org/abs/1904.00827, arXiv:1904.00827.

[4] Ambrus Kaposi and Szumi Xie. Second-order generalised algebraic theories: signatures and first-order semantics, 2024. Draft paper. Available: https://akaposi.github.io/sogat.pdf.