Notions of computability at higher types III

John Longley

April 6, 2001

Note: The current contents of this section are imported from the draft paper [Lon99b]; they are not yet integrated into the present paper.

[Explain motivation. Shift in perspective at this point in the paper.]

Weird features about the preceding account which point to its incompleteness:

- Anti-Church's Theses suggest that we should maybe get away from purely functional settings.
- Effective structures are a bit like morphisms into K_1, but this is not explained by our account. It would be nice to relativize the notion of effectivity a bit more.
- Separation of total and partial? Artificiality of treatment of call-by-name etc. issue.

1 Typed PCAs

We first introduce the notion of typed PCAs, a generalization of the usual notion of PCA. Each typed PCA will be associated with some particular class of types as follows:

Definition 1.1 *A* type world *is any non-empty set T of (names for)* types*, equipped with binary operations $\times$ and $\rightarrow$ for forming product and arrow types.*

Examples 1.2 (i) Let **1** be the singleton type world $\{\star\}$, with $\star \rightarrow \star = \star \times \star = \star$).

(ii) Let $F(\gamma_1, \ldots, \gamma_r)$ be the set of *finite* or *simple* types, freely generated via $\times$ and $\rightarrow$ from the basic type names $\gamma_1, \ldots, \gamma_r$. Important examples are $F(\iota)$ and $F(\iota, o)$, where ι and o are thought of as the types of natural numbers and booleans respectively.

(iii) Richer languages of types, for example as in the recursively-typed language FPC [FP94], also give examples of type worlds.

Definition 1.3 *A* partial combinatory type structure *(or typed PCA) over a type world T is just a family of inhabited sets $(A_t \mid t \in T)$, together with partial* *"application" functions $\cdot_{tu} : A_{t\to u} \times A_t \to A_u$ such that for all types t, u, v there exist elements*

$$\begin{array}{lll} k_{tu} \in A_{t\to u\to t}, & s_{tuv} \in A_{(t\to u\to v)\to(t\to u)\to(t\to v)}, & \\ \mathit{pair}_{tu} \in A_{t\to u\to t\times u}, & \mathit{fst}_{tu} \in A_{t\times u\to t}, & \mathit{snd}_{tu} \in A_{t\times u\to u} \end{array}$$

satisfying the following for all appropriately typed a, b, c. (We suppress mention of the application functions, and write $e\downarrow$ to mean the value of an expression e is defined.)

$$\begin{array}{ll} kab \ = \ a, & sab\downarrow, \\ sabc = (ac)(bc) & \text{whenever } (ac)(bc)\downarrow, \\ \mathit{fst}(\mathit{pair}\ a\ b) = a & \mathit{snd}(\mathit{pair}\ a\ b) = b. \end{array}$$

Note that in the above definition we do not require *extensionality* or anything like it. For some purposes one does not even need the product types in the above definition, but if we are looking for a pleasant mathematical theory they are a very worthwhile investment, as we shall see below.

We have two main classes of motivating examples for the above definition.

Example 1.4 A typed PCA over **1** is just a PCA in the usual sense (with a mild relaxation of the usual condition for s). Note that in this special case it does not matter whether we include the product types in the definition or not, since suitable combinators *pair*, *fst*, *snd* are definable from k, s (see e.g. [Lon95, Chapter 1]).

Example 1.5 A large class of "syntactic" typed PCAs may be obtained from typed programming languages as follows. Let $\mathcal{L}$ be a simply-typed λ-calculus (including product types) over a signature Σ consisting of a set of ground types γ (including the special type ι) and a set of typed constants $c : \sigma$. We will write $\langle -, - \rangle, \mathit{fst}, \mathit{snd}$ for the pairing and projection operations, and write $\mathcal{L}^0_\sigma$ for the set of closed terms of type σ.

Suppose $F = F(\gamma_1, \ldots, \gamma_r)$ where ι is among the γ_i. By a *simply-typed language* $\mathcal{L}$ over $\gamma_1, \ldots, \gamma_r$ let us mean a family of sets $\mathcal{L}_\sigma$ (for $\sigma \in F$) of *terms* of type σ, with the following closure properties:

- If $M \in \mathcal{L}_{\sigma\times\tau}$ then $\mathit{fst}_{\sigma\tau}M \in \mathcal{L}_\sigma$ and $\mathit{snd}_{\sigma\tau}M \in \mathcal{L}_\tau$.
- If $M \in \mathcal{L}_{\sigma\to\tau}$ and $N \in \mathcal{L}_\sigma$ then $MN \in \mathcal{L}_\tau$.

We suppose furthermore that each term M has a set of *free variables* $\mathrm{FV}(M)$, where

$$\mathrm{FV}(\mathit{fst}_{\sigma\tau}M) = \mathrm{FV}(\mathit{snd}_{\sigma\tau}M) = \mathrm{FV}(M), \qquad \mathrm{FV}(MN) = \mathrm{FV}(M) \cup \mathrm{FV}(N).$$

We write $\mathcal{L}^0_\sigma$ for the set $\{M \in \mathcal{L}_\sigma \mid \mathrm{FV}(M) = \emptyset\}$ of *closed* terms of type σ. We also assume we have a notion of *substitution* for terms of $\mathcal{L}$, interacting with free variables in the expected way. Finally, we suppose we are given an *evaluation* function $\mathrm{Eval}_\mathcal{L} : \mathcal{L}^0_\iota \to \mathrm{N}_\perp$.

Now let $(\sim_\sigma \subseteq \mathcal{L}^0_\sigma \times \mathcal{L}^0_\sigma \mid \sigma \in F)$ be any family of equivalence relations compatible with the projections and application, and with the evaluation function. Then the structure $\mathcal{L}^0/\sim = (\mathcal{L}^0_\sigma/\sim_\sigma \mid \sigma \in F)$ is a ...

Then the closed term model for $\mathcal{L}$ (modulo any reasonable equivalence relation containing at least computational equality) is a typed PCA. (Note that for call-by-value languages we need to take just the *terminating* closed terms of $\mathcal{L}$.)

The following is now a straightforward "typed" analogue of the usual definition:

Definition 1.6 *Given a typed PCA A over T, we may define the category of* assemblies *over A as follows.*

(i) An assembly X consists of

- *a type $t(X) \in T$,*
- *an underlying set $|X|$,*
- *for each $x \in |X|$, a non-empty set $\|x\| \subseteq A_{t(X)}$ of* realizers *for x.*

An assembly X is a modest set *if for all $x, y \in |X|$ and $a \in A_{t(X)}$ we have,*

$$a \in \|x\| \wedge a \in \|y\| \;\Rightarrow\; x = y.$$

(ii) A morphism of assemblies $f : X \to Y$ is a function $f : |X| \to |Y|$ such that there exists $r \in At(X)- > t(Y)$ which tracks f in the usual way—that is, for all $x \in |X|$ and $a \in \|x\|$ we have $ra \in \|f(x)\|$.

Assemblies and their morphisms always form a category; we write it as $\mathbf{Asm}(A)$, and write $\mathbf{Mod}(A)$ for the full subcategory of modest sets. [Intuition for this definition: why it took so long!]

As pointed out above, this generalizes the construction of $\mathbf{Asm}(A)$ for A a PCA. The following theorem shows that many of the well-known properties of categories of assemblies over PCAs hold true in this more general setting. The proofs are straightforwardly obtained from the usual proofs by adding types to all the realizers in the only way possible.

Theorem 1.7 *(i) $\mathbf{Asm}(A)$ is locally cartesian-closed.*

(ii) $\mathbf{Asm}(A)$ is a $\nabla\Gamma$-category *in the sense of [Lon95, Section 1.4]: that is, a regular category equipped with exact faithful functors $\Gamma : \mathbf{Asm}(A) \to \mathbf{Set}$ and $\nabla : \mathbf{Set} \to \mathbf{Asm}(A)$ such that $\Gamma \dashv \nabla$ and $\Gamma\nabla \cong id$.*

Remark 1.8 In [Lon95, Section 1.4] it is shown that, for A a PCA, the category $\mathbf{Asm}(A)$ is the free $\nabla\Gamma$-category over $\mathbf{Mod}(A)$ in a certain sense. The proof can be easily adapted to yield the same result for an arbitrary typed PCA.

One can also define the *realizability category* $\mathbf{RC}(A)$ as the exact completion of the regular category $\mathbf{Asm}(A)$. In the untyped case, $\mathbf{RC}(A)$ is just the standard realizability topos over the PCA A.

Note that even when $\mathbf{RC}(A)$ is not a topos, the categories $\mathbf{RC}(A)$, $\mathbf{Asm}(A)$, $\mathbf{Mod}(A)$. all have enough structure to allow us to interpret first-order logic. (Actually, we don't yet have disjunction since $\mathbf{RC}(A)$ does not automatically have binary coproducts. But under a mild extra condition that we can "simulate" the booleans within A, it does and we can model disjunction.) In the case of term models for typed programming languages, the interpretation of first-order logic in these categories is exactly the *typed realizability* interpretation discussed in [Lon99a]. [Expand this!]

1.1 Applicative morphisms between typed PCAs

In [Lon95, Chapter 2] we introduced and studied the notion of *applicative morphism* between PCAs, and the corresponding functors between realizability categories. We now show that all this theory lifts in a straightforward way to the typed setting.

Definition 1.9 *Let A be a typed PCA over T, and B a typed PCA over U.*

(i) An applicative morphism *(f, γ) from A to B consists of*

- *a function $f : T \to U$ (no preservation properties required); and*
- *for each $t \in T$, a total relation γ_t from A_t to B_{ft}*

such that for all $t, u \in T$ there is an element $r \in B_{f(t \to u) \to f(t) \to f(u)}$ which "realizes" $\cdot_{tu}$, in the sense that

$$(\gamma_{t \to u}(a, a') \wedge \gamma_t(b, b') \wedge ab \text{ defined}) \implies \gamma_u(ab, ra'b').$$

(ii) If (f, γ) and (g, δ) are applicative morphisms $A \to B$, we say there is an applicative transformation *from f to g if for each $t \in T$ there exists $r \in B_{f(t) \to g(t)}$ such that*

$$\gamma_t(a, a') \implies \delta_t(a, ra').$$

(If an applicative transformation $f \to g$ exists it is unique.) We write **TPCA** *for the 2-category of typed PCAs, applicative morphisms and applicative transformations.*

Once again, the verification that this indeed defines a 2-category is an easy typed adaptation of the proof in [Lon95].

Example 1.10 Applicative morphisms between typed PCAs over $\{\star\}$ are exactly applicative morphisms in the original sense. Several examples of applicative morphisms were discussed in [Lon95]; in addition, Lietz [Lie99] has considered applicative morphisms between $\mathcal{P}\omega$ and K_2.

Example 1.11 In [Lon98] we defined a notion of *simulation* between extensional partial type structures (essentially, these are extensional total typed PCAs over the simple types over $N_\perp$). A simulation $\gamma : A \to B$ is a family of total relations $\gamma_t : A_t \to B_t$ such that γ_0 is the identity on $N_\perp$, and such that

$$\gamma_{t \to u}(f, g) \wedge \gamma_t(x, y) \implies \gamma_u(fx, gy).$$

Clearly, such simulations are just a special kind of applicative morphism between extensional PTSs.

Example 1.12 Likewise, in [Lon99a] we defined a *translation* θ between (simply-)typed programming languages $\mathcal{L}$ and $\mathcal{L}'$ to be a family of functions θ_σ from $\mathcal{L}$-terms of type σ to $\mathcal{L}'$-terms of type σ that respect free variables and commute with projections, application, and evaluation of ground type terms. If $\mathcal{L}^0/\sim$ and $\mathcal{L}'^0/\sim'$ are closed term models (viewed as typed PCAs), θ induces an applicative morphism (f,γ) between them, where f is the identity on types, and γ is given by all instances of of the formula $\gamma([M],[\theta M])$. Thus, translations are again a special kind of applicative morphism.

Exactly as in the untyped case, applicative morphisms give rise to functors between the categories of assemblies, and moreover one can characterize precisely the functors that arise in this way. Once again, the results and proofs are simple adaptations of those in [Lon95]—however, we summarize the proofs here as they have not been published elsewhere.

Definition 1.13 *Let $(\mathcal{C},\Gamma_C,\nabla_C)$ and $(\mathcal{D},\Gamma_D,\nabla_D)$ be $\nabla\Gamma$-categories. A* $\nabla\Gamma$-functor *from $\mathcal{C}$ to $\mathcal{D}$ is an exact functor $F:\mathcal{C}\to\mathcal{D}$ such that $\Gamma_D F\cong\Gamma_C$ and $F\nabla_C\cong\nabla_D$. We write $\nabla\Gamma\mathbf{Reg}$ for the 2-category of $\nabla\Gamma$-categories, $\nabla\Gamma$-functors and all natural transformations between them.*

It is shown in [Lon95, Proposition 1.4.4] that any $\nabla\Gamma$-functor automatically preserves the unit of the adjunction $\Gamma\dashv\nabla$ modulo the obvious isomorphisms.

Theorem 1.14 *(i) Applicative morphisms $A\to B$ give rise in a functorial way to $\nabla\Gamma$-functors $\mathbf{Asm}(A)\to\mathbf{Asm}(B)$. Similarly, applicative transformations give rise to natural transformations between such functors. Thus, the construction $\mathbf{Asm}(-)$ extends to a 2-functor from* **TPCA** *to $\nabla\Gamma\mathbf{Reg}$.*

(ii) Moreover, the 2-functor $\mathbf{Asm}(-)$ is locally an equivalence*: that is, for any typed PCAs A,B the functor*

$$\mathbf{Asm}:\mathbf{TPCA}[A,B]\;\longrightarrow\;\nabla\Gamma\mathbf{Reg}[\mathbf{Asm}(A),\mathbf{Asm}(B)]$$

is part of an equivalence of categories.

[Sketch proof.]
[What morphisms are from the weak CCC point of view.]

1.2 Special morphisms and equivalences

Special classes: discrete, projective, decidable; typeful (i.e. preserving application). Adjunctions/equivalences. General results on this 2-category (e.g. extensional TSs form a poset; retractions are typeful; K_1 is initial in some sense). Typeful morphisms as those commuting with inclusion of some minimal TS (Mike's idea).

We may now introduce one of the central concepts of the paper.

Definition 1.15 *If A,B are typed PCAs, we say A,B are* applicatively equivalent *if they are equivalent in the 2-category* **TPCA**.

The following is an easy corollary of Theorem 1.14.

Theorem 1.16 *A, B are applicatively equivalent iff $\mathbf{Asm}(A), \mathbf{Asm}(B)$ are equivalent as categories.*

[Proof?]

[Intuition behind equivalences. General results, e.g. automatically typeful (??)]

Examples like $\mathcal{P}\omega$ and D_∞ (Bauer). Engeler models. The general fact behind this. Call-by-name/call-by-value/lazy: general fact, term model examples.

[Possible material:] Some easy equivalences (e.g. for extensional things?): Sufficiency of pure types. Different treatments of partiality: interderivability (via retractions) and equivalences, e.g. of CBN/CBV; term model examples. Sufficiency of total functionals over $N_\perp$ to represent partial TSs. (Partial CCCs?)

["Semi-equivalences" suffice for LFA. Hierarchy of "goodness of fit" criteria.]

1.3 The Lietz-Streicher theorem

We have seen that for any typed PCA A we get a realizability category $\mathbf{RC}(A)$ with pretty good structure. The above results also show that, even if A isn't given as a PCA, it's quite often equivalent to one, and so $\mathbf{RC}(A)$ quite often turns out to be a topos. The crucial property of A that makes this happen appears to be the existence of some kind of "universal" type t such that the whole of A can in some sense be reduced to type t. This accords with the intuition that the reason we get a topos in the untyped case is because we have a small "universe" which allows for impredicativity.

Indeed, shortly after seeing an early draft of this paper, Lietz and Streicher have obtained the following beautiful theorem characterizing those realizability models that are equivalent to one given by a PCA:

Theorem 1.17 *For any typed PCA A, the following are equivalent:*

(i) A is applicatively equivalent to a PCA.

(ii) A contains a type u (called a universal type*) such that for any type t in A there exist elements $a \in A_{t\to u}$ and $b \in A_{u\to t}$ such that for all $x \in A_t$ we have $b(ax) = x$.*

(iii) $\mathbf{Asm}(A)$ *contains a generic mono (that is, a mono of which all other monos are pullbacks).*

(iv) $\mathbf{RC}(A)$ *is a topos.*

The significance of this result, as we see it, ...

1.4 Some constructions on typed PCAs

Extensional collapse [Zucker] (w.r.t. arbitrary PER on ground type); examples. Non-functoriality; stress that it's not the only way in general! Greediness... Stable under equivalences?? (Starting from an extensional structure get the same one. Iterations? extensional realizability?) "Modifying" construction;

relationship with modified realizability. (Partial version: doesn't do much.) Composing EC and Modify in various ways. General results? Type-respecting morphisms: sufficient conditions?

2 Examples of typed PCAs

We can now revisit the examples of matching pairs of typed and untyped realizability models in my paper, and see that in many cases we simply have an applicative equivalence between the closed term model of the typed programming language (modulo observational equivalence) and the corresponding PCA. It is easy to see that this equivalence implies logical full abstraction in the sense of [Lon99a], since both typed PCAs give rise to the same category of assemblies, and both realizability interpretations are just the interpretation of first-order logic in this category. (In fact, logical full abstraction is weaker than applicative equivalence, and seems to correspond something like the fact that the relevant *sub-logoses* of the two categories of assemblies are equivalent...)

Example 2.1 PCF^{++} *(i.e. PCF+parallel-or+exists) and* $T^{\omega}_{\mathrm{rec}}$*. Let* A *be the closed term model for* PCF^{++} *(modulo obs.eq.), and let* $B = T^{\omega}_{re}$*. Since* $T^{\omega}_{\mathrm{rec}}$ *can be simulated by the* PCF^{++} *type* $\iota \to \iota$ *(or even* $\iota \to o$*), we have an applicative morphism* $\delta : B \to A$*. Moreover, since in the effective Scott model all the* PCF^{++} *types are retracts of* $T^{\omega}_{\mathrm{rec}}$ *(see [Plo78]), we have an applicative morphism* $\gamma : B \to A$*. The isomorphism* $\delta\gamma \cong \mathrm{id}_A$ *comes from the fact that the required embeddings/retractions are definable in* PCF^{++}*; the isomorphism* $\gamma\delta \cong \mathrm{id}_B$ *is fairly trivial. So "realizability over* PCF^{++}*" and "realizability over* $T^{\omega}_{\mathrm{rec}}$*" amount to exactly the same thing. In particular,* $\mathbf{RC}(\mathrm{PCF}^{++})$ *is a topos!*

Example 2.2 *PCF+*H *and* $\mathcal{B}_{2\mathrm{rec}}$*. Exactly the same holds, because of the universality of type 2 in the effective SR functionals (see [Lon98]).*

Example 2.3 *What about PCF and the untyped model* L_U *of Streicher et al. [MRS99]? Well, we can't simulate* L_U *within PCF in the appropriate sense, because PCF doesn't possess a universal type (folklore; consequence of games models). In fact, for this reason, it seems pretty clear that* $\mathbf{RC}(\mathrm{PCF})$ *won't be a topos, and hence PCF can't be applicatively equivalent to any PCA. However, the language FPC (morally, PCF with recursive types) does contain the required universal type which allows us to simulate* L_U*, and so FPC and* L_U *are applicatively equivalent.*

(This last example shows, incidentally, that applicative equivalence is strictly stronger than logical full abstraction.)

Now the non-functional languages:

Example 2.4 *PCF+*catch *and* $\mathcal{B}_{\mathrm{rec}}$ *(probably the best example so far). To simulate PCF+*catch *in* $\mathcal{B}_{\mathrm{rec}}$*, we use the fact that* $PCF +$ catch *has an interpretation in effective sequential algorithms, and that* $\mathcal{B}_{\mathrm{rec}}$ *is universal for a sufficiently large part of the effective sequential algorithms model (see [Lon98]).*

*Simulating $\mathcal{B}_{\mathrm{rec}}$ in type $\iota \to \iota$ of PCF+*catch *is easy. So we have applicative morphisms both ways; we want to show that these constitute an equivalence. The interesting bit is to show that for any type t there's a PCF+*catch *term*

$$\textit{algorithm-for} : t \to (\iota \to \iota)$$

which extracts from any term its underlying sequential algorithm. But this is exactly the fact that is used to show logical full abstraction in [Lon99a].

Note that PCF+catch here could be replaced by μPCF, or indeed by a fragment of SML including (somewhat constrained) uses of exceptions and even references. (We claim that these languages are all applicatively equivalent to one another and hence to $\mathcal{B}_{\mathrm{rec}}$.) Thus, realizability over this fragment of SML yields exactly the topos $\mathbf{RT}(\mathcal{B}_{\mathrm{rec}})$.

Example 2.5 *What about PCF+*timeout *and K_2? Although K_2 is logically fully abstract for PCF+*timeout*, and there are applicative morphisms both ways, it seems that these do not quite constitute an equivalence. (The reason is rather technical.) However, we believe that by slightly modifying the definition of PCF+*timeout *it should be possible to obtain an applicative equivalence, but we have not done this yet. [REVISE THIS!]*

Example 2.6 *What about PCF+*quote *and K_1? It is not immediately clear how one should make the language PCF+*quote *into a typed PCA, since K_1 can be appropriately simulated by the type nat. [Sorry, don't understand this!] (In fact, PCF+*quote *is a slightly dishonest language, since once we have* quote *around, our language might as well be untyped!)*

[Examples coming from *total* type structures?]

2.1 Further thoughts and questions

It is also natural to look for other applicative equivalences between typed and untyped structures. One can either ask whether there is a PCA that is equivalent to some known typed language, or a typed language that is equivalent to some known PCA. Even if there's no PCA equivalent to a given language, one can still ask for a "semantic" characterization of some equivalent typed PCA. Something I'd particularly like to find (for personal reasons) is a mathematically natural typed PCA that's equivalent to a fragment of SML with exceptions including WILDCARD exception handlers.

The research programme that's implicit here is to identify what seem to be the "natural" notions of computability/realizability (as embodied by typed PCAs up to applicative equivalence). Of course, we also want to understand what morphisms exist between these natural notions. It is immediate, for example, that a translation between typed languages will induce an applicative morphism between corresponding PCAs (if these exist). The diagram below can be seen as a stab at what this big picture might look like.

$$\begin{tikzcd}
 & \text{PCF}+\mathsf{quote}\ (K_1) & \\
 & \text{PCF}+\mathsf{catch}+\mathsf{timeout}\ (?K_{2\mathrm{rec}}) \arrow[u] & \\
\text{PCF}+\mathsf{timeout} \arrow[ur] & & \text{PCF}+\mathsf{catch}\ (\mathcal{B}_{\mathrm{rec}}) \arrow[ul] \\
\text{PCF}^{++}\ (T^{\omega}_{\mathrm{rec}}) \arrow[u] & & \text{PCF}+\mathsf{H}\ (\mathcal{B}_{2\mathrm{rec}}) \arrow[u] \arrow[ull] \\
 & \text{PCF}\ \ (L_U) \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

[Mention relative realizability]

3 Related areas and further directions

Afterword

Tribute to Kleene.

References

[FP94] M.P. Fiore and G.D. Plotkin. An axiomatisation of computationally adequate domain theoretic models of FPC. In *Proc. 9th Annual Symposium on Logic in Computer Science*. IEEE, 1994.

[Lie99] P. Lietz. Comparing realizability over $\mathcal{P}\omega$ and K_2. Draft paper, 1999.

[Lon95] J.R. Longley. *Realizability Toposes and Language Semantics.* PhD thesis, University of Edinburgh, 1995. Available as ECS-LFCS-95-332.

[Lon98] J.R. Longley. The sequentially realizable functionals. Technical Report ECS-LFCS-98-402, Department of Computer Science, University of Edinburgh, 1998. To appear in *Annals of Pure and Applied Logic.*

[Lon99a] J.R. Longley. Matching typed and untyped realizability. In L. Birkedal, J. van Oosten, G. Rosolini, and D.S. Scott, editors, *Proc. Workshop on Realizability, Trento*, 1999. Published as Electronic Notes in Theoretical Computer Science 23 No. 1, Elsevier. Available via `http://www.elsevier.nl/locate/entcs/volume23.html`.

[Lon99b] J.R. Longley. Unifying typed and untyped realizability. Electronic note, available at `http://www.dcs.ed.ac.uk/home/jrl/unifying.txt`., 1999.

[MRS99] M. Marz, A. Rohr, and T. Streicher. Full abstraction and universality via realisability. In *Proc. 14th Annual IEEE Symposium on Logic in Computer Science*. IEEE, 1999.

[Plo78] G.D. Plotkin. T^ω as a universal domain. *Journal of Computer and System Sciences*, 17:209–236, 1978.