A Type-Theoretic Model of Hierarchical Modules

Andreas Abel

4 May 2019

Abstract

1 Introduction

2 Hierarchical Modules: A Minimal Calculus

Disclaimer: may be outdated, the next section is actively worked on.

Abstract syntax.

s	::=	$\mathsf{module}\ x\ s^*$	module definition
	|	$\mathsf{ref}\ x^+$	module reference
x			simple name
q	::=	x^+	qualified name
α, β	::=	s^*	module content
Γ, Δ	::=	α^*	scope

A concrete syntax for statements could be the Agda module syntax, i. e., `module` x `where` s^* for $\mathsf{module}\ x\ s^*$, and `module _ =` q for $\mathsf{ref}\ q$.

Sequences of type $\_^*$ may be empty whereas $\_^+$ stands for non-empty sequences.
We write ε for the empty sequence and use a dot or raised dot for concatenation.
We silently cast elements to singleton sequences, e. g., we write x for $x.\varepsilon$.

Judgements.

$\Gamma \ni q$	Scope Γ declares name q
$\Gamma \vdash q$	Scope Γ declares name q (disambiguated)
$\Gamma \vdash \alpha$	Module content α is well-scoped in Γ
$\vdash \Gamma$	Scope Γ is well-formed

$\boxed{\Gamma \ni q}$

$$\textsf{here}\ \frac{}{\Gamma \cdot \alpha \cdot \textsf{module}\ x\ \beta \ni x} \qquad \textsf{there}\ \frac{\Gamma \cdot \alpha \ni q}{\Gamma \cdot \alpha \cdot s \ni q}$$

$$\textsf{into}\ \frac{\beta \ni q}{\Gamma \cdot \alpha \cdot \textsf{module}\ x\ \beta \ni x.q} \qquad \textsf{parent}\ \frac{\Gamma \ni q}{\Gamma \cdot \alpha \ni q}$$

The judgement $\Gamma \ni q$ is a decidable set, its inhabitants are paths to the definition site of a name. $\Gamma \ni q$ may have more than one inhabitant, meaning that name q would be ambiguous. We can disambiguate by always taking the closest declaration as the definition of a name.

$\boxed{\Gamma \vdash q}$ (implies $\Gamma \ni q$).

$$\textsf{here}\ \frac{}{\Gamma \cdot \alpha \cdot \textsf{module}\ x\ \beta \vdash x} \qquad \textsf{there}\ \frac{\Gamma \cdot \alpha \vdash q \qquad s \not\ni q}{\Gamma \cdot \alpha \cdot s \vdash q}$$

$$\textsf{into}\ \frac{\beta \vdash q}{\Gamma \cdot \alpha \cdot \textsf{module}\ x\ \beta \vdash x.q} \qquad \textsf{parent}\ \frac{\Gamma \vdash q \qquad \alpha \not\ni q}{\Gamma \cdot \alpha \vdash q}$$

$\Gamma \vdash q$ is a proposition, i. e., contains at most one inhabitant. This inhabitant, when it exists, may be the considered the *resolution* of name q in Γ, and it is the equivalent of a de Bruijn index in lambda calculus.

Lemma 1 (Unambiguity of resolved names)**.** *If* $y, z : (\Gamma \vdash q)$ *then* $y = z$.

Proof. By induction on $y, z : (\Gamma \vdash q)$. For a representative case, consider $y = \textsf{into}\ y'$ and $z = \textsf{there}\ z'$. Assuming the last declaration in Γ is $s = \textsf{module}\ x\ \beta$, this means that $y' : (\beta \vdash q)$, but also $s \not\ni q$, implied by z. However, $\textsf{into}\ y' : (s \vdash q)$, which is impossible, as it implies $s \ni q$. □

Well-scopedness of statement (lists): propositions $\boxed{\Gamma \vdash s}$ and $\boxed{\Gamma \vdash \alpha}$.

$$\frac{\Gamma \vdash q}{\Gamma \vdash \textsf{ref}\ q} \qquad \frac{\Gamma \perp x \qquad \Gamma \vdash \beta}{\Gamma \vdash \textsf{module}\ x\ \beta} \qquad \frac{}{\Gamma \vdash \varepsilon} \qquad \frac{\Gamma \vdash \alpha \qquad \Gamma \cdot \alpha \vdash s}{\Gamma \vdash \alpha \cdot s}$$

Declaring a new name x involves checking for name clashes via judgement $\Gamma \perp x$. We can have $\Gamma \perp x$ vacuously true without introducing ambiguity, but this might permit unwanted shadowing. For instance, content $\textsf{module}\ x\ \alpha \cdot \textsf{module}\ x\ \beta \cdot \textsf{ref}\ x$ may be ruled out since it introduces the same name x *on the same level* twice. The reference x is still unambiguous as we resolve it to the last definition of x, however, this is likely to confuse programmers working in a language with such shadowing. In contrast, $\textsf{module}\ x\ (\textsf{module}\ x\ \varepsilon)$ which declares a parent x with a child named x, is less controversial, since the shadowing is on different levels. It is very common that local definitions may share global definitions, for instance.

To disallow shadowing on the same level, we define proposition $\boxed{\Gamma \perp x}$ as follows, using auxiliary proposition $\boxed{\alpha \perp x}$ and $\boxed{s \perp x}$.

$$\frac{}{\varepsilon \perp x} \qquad \frac{\alpha \perp x}{\Gamma \cdot \alpha \perp x} \qquad \frac{\alpha \perp x \qquad s \perp x}{\alpha \cdot s \perp x} \qquad \frac{}{\mathsf{ref}\ q \perp x} \qquad \frac{x \neq x'}{\mathsf{module}\ x'\ \beta \perp x}$$

Note that for $\Gamma \cdot \alpha \perp x$, we only check the last content block α, which is on the same level as the to-be-defined name x. However, then we need to check *all* statements within that block α.

Well-formedness of scopes $\vdash \Gamma$.

$$\frac{}{\vdash \varepsilon} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash \alpha}{\vdash \Gamma \cdot \alpha}$$

3 Hierarchical Modules: A Minimal Calculus With Aliases

Disclaimer: cut-and-paste from previous section, repetitive.

Abstract syntax.

s	::=	$\mathsf{module}\ x\ s^*$	module definition
	$\mid$	$\mathsf{alias}\ x\ q$	module alias
x			simple name
q	::=	x^+	qualified name
α, β	::=	s^*	module content
Γ, Δ	::=	α^*	scope

A concrete syntax for statements could be the Agda module syntax, i. e., `module` x `where` s^* for $\mathsf{module}\ x\ s^*$, and `module` x `=` q for $\mathsf{alias}\ x\ q$.

Another interpretation would be a file system with folders only. Then $\mathsf{module}\ x\ s^*$ defines a folder with subfolders s^*, and $\mathsf{alias}\ x\ q$ is a symbolic link to q named x. However, file systems usually allow arbitrary recursion, e. g., we can have folders A and B where A contains a link to B and vice versa. This would yield an infinite path $A.B.A.B....$ In contrast, we are aiming at well-founded structures, where pathes are finite and aliases can in principle be expanded.

Sequences of type $\_^*$ may be empty whereas $\_^+$ stands for non-empty sequences. We write ε for the empty sequence and use a dot or raised dot for concatenation. We silently cast elements to singleton sequences, e. g., we write x for $x.\varepsilon$. The name x defined in statement s is $\boxed{\mathsf{def}\ s}$, thus, $\mathsf{def}(\mathsf{module}\ x\ \beta) = \mathsf{def}(\mathsf{alias}\ x\ q) = x$.

Judgements.

$\Gamma \ni q$	Scope Γ declares name q
$\Gamma \vdash q$	Scope Γ declares name q (disambiguated)
$\Gamma \vdash \alpha$	Module content α is well-scoped in Γ
$\vdash \Gamma$	Scope Γ is well-formed

$\boxed{\Gamma \ni q}$

$$\textsf{here}\ \frac{}{\Gamma \cdot \alpha \cdot \textsf{module}\ x\ \beta \ni x} \qquad \textsf{there}\ \frac{\Gamma \cdot \alpha \ni q}{\Gamma \cdot \alpha \cdot s \ni q} \qquad \textsf{parent}\ \frac{\Gamma \ni q}{\Gamma \cdot \alpha \ni q}$$

$$\textsf{into}\ \frac{\beta \ni q}{\Gamma \cdot \alpha \cdot \textsf{module}\ x\ \beta \ni x.q} \qquad \textsf{follow}\ \frac{\Gamma \cdot \alpha \ni q'.q}{\Gamma \cdot \alpha \cdot \textsf{alias}\ x\ q' \ni x.q}$$

The judgement $\Gamma \ni q$ is a decidable set, its inhabitants are paths to the definition site of a name.

To resolve a qualified name $x.q$ matching an alias q, $\Gamma \cdot \textsf{alias}\ x\ q' \ni x.q$, the intuitive procedure would be to look up the content α of the aliased module q', and then resolve q' in α. However, the decidability would be less trivial; for termination, we would have to argue that α is structurally smaller than Γ. While this is easy to see, it still would be more laborious to formalize: we would need a relation that maps a name to its content (if any), define a *smaller-than* relation on scopes, prove its well-foundedness, and prove that in scope Γ, the retrieved content α is always smaller than Γ. Our follow rule instead expands the alias on the fly, and continues with the expanded name.

For fixed Γ and q, the set $\Gamma \ni q$ may have more than one inhabitant, meaning that name q would be ambiguous. We can disambiguate by always taking the closest declaration as the definition of a name.

$\boxed{\Gamma \vdash q}$ (implies $\Gamma \ni q$).

$$\textsf{here}\ \frac{}{\Gamma \cdot \alpha \cdot \textsf{module}\ x\ \beta \vdash x} \qquad \textsf{there}\ \frac{\Gamma \cdot \alpha \vdash q \qquad s \not\ni q}{\Gamma \cdot \alpha \cdot s \vdash q} \qquad \textsf{parent}\ \frac{\Gamma \vdash q \qquad \alpha \not\ni q}{\Gamma \cdot \alpha \vdash q}$$

$$\textsf{into}\ \frac{\beta \vdash q}{\Gamma \cdot \alpha \cdot \textsf{module}\ x\ \beta \vdash x.q} \qquad \textsf{follow}\ \frac{\Gamma \cdot \alpha \vdash q'.q}{\Gamma \cdot \alpha \cdot \textsf{alias}\ x\ q' \vdash x.q}$$

$\Gamma \vdash q$ is a proposition, i. e., contains at most one inhabitant. This inhabitant, when it exists, may be the considered the *resolution* of name q in Γ, and it is the equivalent of a de Bruijn index in lambda calculus.

Lemma 2 (Unambiguity of resolved names)**.** *If* $y, z : (\Gamma \vdash q)$ *then* $y = z$.

Proof. By induction on $y, z : (\Gamma \vdash q)$. For a representative case, consider $y = \textsf{into}\ y'$ and $z = \textsf{there}\ z'$. Assuming the last declaration in Γ is $s = \textsf{module}\ x\ \beta$, this means that $y' : (\beta \vdash q)$, but also $s \not\ni q$, implied by z. However, $\textsf{into}\ y' : (s \vdash q)$, which is impossible, as it implies $s \ni q$. □

Well-scopedness of statement (lists): propositions $\boxed{\Gamma \vdash s}$ and $\boxed{\Gamma \vdash \alpha}$.

$$\frac{\Gamma \perp x \qquad \Gamma \vdash q}{\Gamma \vdash \mathsf{alias}\ x\ q} \qquad \frac{\Gamma \perp x \qquad \Gamma \vdash \beta}{\Gamma \vdash \mathsf{module}\ x\ \beta} \qquad \frac{}{\Gamma \vdash \varepsilon} \qquad \frac{\Gamma \vdash \alpha \qquad \Gamma \cdot \alpha \vdash s}{\Gamma \vdash \alpha \cdot s}$$

Declaring a new name x involves checking for name clashes via judgement $\Gamma \perp x$. We can have $\Gamma \perp x$ vacuously true without introducing ambiguity, but this might permit unwanted shadowing. For instance, content $\mathsf{module}\ x\ \alpha \cdot \mathsf{module}\ x\ \beta \cdot \mathsf{alias}\ y\ x$ may be ruled out since it introduces the same name x *on the same level* twice. The reference x is still unambiguous as we resolve it to the last definition of x, however, this is likely to confuse programmers working in a language with such shadowing. In contrast, $\mathsf{module}\ x\ (\mathsf{module}\ x\ \varepsilon)$ which declares a parent x with a child named x, is less controversial, since the shadowing is on different levels. It is very common that local definitions may share global definitions, for instance.

To disallow shadowing on the same level, we define proposition $\boxed{\Gamma \perp x}$ as follows:

$$\frac{}{\varepsilon \perp x} \qquad \frac{\alpha \not\vdash x}{\Gamma \cdot \alpha \perp x}$$

Note that for $\Gamma \cdot \alpha \perp x$, we only check the last content block α, which is on the same level as the to-be-defined name x. (However, we then check *all* statements within that block α.)

Well-formedness of scopes $\vdash \Gamma$.

$$\frac{}{\vdash \varepsilon} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash \alpha}{\vdash \Gamma \cdot \alpha}$$

4 Opening Modules

Module systems often feature statements that let us include the contents of one module in another. In Agda, this is the `open` statement.

$$s \quad ::= \quad \mathsf{open}\ q \qquad \text{import content of module } q$$

In the most basic form, where $\mathsf{open}\ q$ imports the *whole* content of module q into the current scope, open can be realized analogously to alias:

$$\mathsf{open}\ \frac{\Gamma \ni q'.q}{\Gamma \cdot \mathsf{open}\ q' \ni q} \qquad \mathsf{open}\ \frac{\Gamma \vdash q'.q}{\Gamma \cdot \mathsf{open}\ q' \vdash q} \qquad \frac{\Gamma \vdash q}{\Gamma \vdash \mathsf{open}\ q}$$

In fact, open is just an alias with an empty name. We might consider aliases that define *qualified* names, including the empty name.

s	::=	$\mathsf{alias}\ xs\ q$	import content of module q under extra qualification xs
xs	::=	x^*	qualification prefix

Then $\mathsf{open}\ q = \mathsf{alias}\ \varepsilon\ q$.

Setting $\mathsf{def}(\mathsf{alias}\ xs\ q) = xs$ updates the here rule. The follow rule is virtually unchanged, just generalized from x to xs.

$$\mathsf{follow}\ \frac{\Gamma \ni q'.q}{\Gamma \cdot \mathsf{alias}\ xs\ q' \ni xs.q} \qquad \mathsf{follow}\ \frac{\Gamma \vdash q'.q}{\Gamma \cdot \mathsf{alias}\ xs\ q' \vdash xs.q}$$

It is a bit unclear though what the shadowing rules for aliases xs should be.

The simulation of open via a generalized alias can be reversed: we can encode $\mathsf{alias}\ x\ q$ as $\mathsf{module}\ x\ (\mathsf{open}\ q)$. Similarily, a generalized $\mathsf{alias}\ x_1 \ldots x_n\ q$ where $n \geq 0$ can be encoded by $\mathsf{module}\ x_1\ (\ldots \mathsf{module}\ x_n\ (\mathsf{open}\ q))$. It seems thus that open is more primitive than alias, and we can drop alias from the core module calculus.

4.1 Ambiguity

The names brought into scope by open might clash with names already in scope. Thus with open, the judgement $\Gamma \vdash q$ is no longer a proposition in general. However, whenever we *use* a name q, we require it to be non-ambiguous.

4.2 Import restriction

It is very common that imports can be specified precisely, i. e., we may only import certain identifiers from a module. Agda and Haskell have import directives for selecting only certain identifiers (Agda keyword `using`) or selecting all but certain identifiers (Agda and Haskell keyword `hiding`). Agda also allows `renaming` of identifiers. These directives can be subsumed under a *renaming map* ρ that maps simple names to simple names. The imported symbols are the domain of ρ, their names are given by the range of ρ. The map should be injective, i. e., two different names should not be mapped to the same name. It need not necessarily be functional, i. e., nothing speaks against importing a name twice under different names.[1] It might thus be more elegant to consider ρ as a map from names defined by the import to their origin. Thus, its range consists of the imported symbols, and its domain gives their new names.

$$s \quad ::= \quad \mathsf{open}\ q\ \rho \qquad \text{import content, as given by } \rho\text{, of module } q$$

[1] Agda forbids duplicate imports of the same symbol (even when under different name) in a single `open` statement. As a consequence ρ, can be stored as a finite map from imported symbols to their new name. One may want to argue in favor of Agda that importing the same symbol twice (while under different names) is non-sensical and could be errorneous and unintended by the user. Agda does allow repetitive import of the same symbol in separate `open` statements, though. One may want to argue further that this way, the user states more explicitely that they want to import the same symbol twice. However, one may argue just the other way round: a second `open` statement with the same explicit import is given in error, and stating the duplicate import twice in the same `open` statement emphasizes the intent of the user.

When trying to resolve a qualified name $x.xs$ in $\mathsf{open}\ q\ \rho$, we subject x to the renaming ρ.

$$\mathsf{open}\ \frac{\Gamma \ni q.\rho(x).xs}{\Gamma \cdot \mathsf{open}\ q\ \rho \ni x.xs} \qquad \mathsf{open}\ \frac{\Gamma \vdash q.\rho(x).xs}{\Gamma \cdot \mathsf{open}\ q\ \rho \vdash x.xs}$$

$$\frac{\Gamma \vdash q \qquad \forall x : \mathsf{rng}\,\rho.\ \Gamma \vdash q.x}{\Gamma \vdash \mathsf{open}\ q\ \rho}$$

An open statement is well-formed if it does not try to import non-existing symbols. In Agda-2.6.0, a violation of this sanity condition is not a hard error, it only produces a warning.

5 Discussion

6 Related Work

Acknowledgments