λ-bound variables in ML cannot be used polymorphically within a function abstraction

For example, $\lambda f\,((f\,\texttt{true}) :: (f\,\texttt{nil}))$ and $\lambda f\,(f\,f)$ are not typeable in the Mini-ML type system.

λ-bound variables in ML cannot be used polymorphically within a function abstraction

For example, $\lambda f\,((f\,\texttt{true}) :: (f\,\texttt{nil}))$ and $\lambda f\,(f\,f)$ are not typeable in the Mini-ML type system.

Syntactically, because in rule

$$(\mathbf{fn})\ \frac{\Gamma, \boxed{x : \tau_1} \vdash M : \tau_2}{\Gamma \vdash \lambda x\,(M) : \tau_1 \to \tau_2}$$

the abstracted variable has to be assigned a *trivial* type scheme (recall $x : \tau_1$ stands for $x : \forall\{\,\}\,(\tau_1)$).

$$\dfrac{\dfrac{}{\{f:\forall\{\}\tau_2\}\vdash f:\tau_4}\,(\mathrm{var}{>}) \qquad \dfrac{}{\{f:\forall\{\}\tau_2\}\vdash f:\tau_5}\,(\mathrm{var}{>})}{\dfrac{\{f:\forall\{\}\tau_2\}\vdash ff:\tau_3}{\{\}\vdash \lambda f(ff):\tau_1}\,(\mathrm{lam})}\,(\mathrm{app})$$

$$\textcircled{4}\ \dfrac{\textcircled{3}\ \dfrac{\textcircled{1}\ \dfrac{}{\{f : \forall\{\}\tau_2\} \vdash f : \tau_4}\,(var\succ) \qquad \textcircled{2}\ \dfrac{}{\{f : \forall\{\}\tau_2\} \vdash f : \tau_5}\,(var\succ)}{\{f : \forall\{\}\tau_2\} \vdash ff : \tau_3}\,(app)}{\{\} \vdash \lambda f(ff) : \tau_1}\,(lam)$$

$\textcircled{1}$ $\forall\{\}\tau_2 > \tau_4$

$\textcircled{2}$ $\forall\{\}\tau_2 > \tau_5$

$\textcircled{3}$ $\tau_4 = \tau_5 \to \tau_3$

$\textcircled{4}$ $\tau_1 = \tau_2 \to \tau_3$

$$\textcircled{4}\ \dfrac{\textcircled{3}\ \dfrac{\textcircled{1}\ \dfrac{}{\{f:\forall\{\}\tau_2\}\vdash f:\tau_4}(\text{var}>) \qquad \textcircled{2}\ \dfrac{}{\{f:\forall\{\}\tau_2\}\vdash f:\tau_5}(\text{var}>)}{\{f:\forall\{\}\tau_2\}\vdash ff:\tau_3}(\text{app})}{\{\}\vdash \lambda f(ff):\tau_1}(\text{lam})$$

$\textcircled{1}$ $\forall\{\}\tau_2 > \tau_4$ so $\tau_2 = \tau_4$

$\textcircled{2}$ $\forall\{\}\tau_2 > \tau_5$ so $\tau_2 = \tau_5$

$\textcircled{3}$ $\tau_4 = \tau_5 \to \tau_3$

$\textcircled{4}$ $\tau_1 = \tau_2 \to \tau_3$

$$\dfrac{\dfrac{①}{\{f:\forall\{\}\tau_2\}\vdash f:\tau_4}\,(var>) \qquad \dfrac{②}{\{f:\forall\{\}\tau_2\}\vdash f:\tau_5}\,(var>)}{\dfrac{\{f:\forall\{\}\tau_2\}\vdash ff:\tau_3}{\{\}\vdash \lambda f(ff):\tau_1}\,(lam)\ ④}\,(app)\ ③$$

① $\forall\{\}\tau_2 > \tau_4$ so $\tau_2 = \tau_4$

② $\forall\{\}\tau_2 > \tau_5$ so $\tau_2 = \tau_5$

③ $\tau_4 = \tau_5 \to \tau_3$

④ $\tau_1 = \tau_2 \to \tau_3$

$\Rightarrow \tau_2 = \tau_2 \to \tau_3$ ✗

No such τ_2 & τ_3 can exist (by counting $\to$ symbols on LHS & RHS of the equation).

λ-bound variables in ML cannot be used polymorphically within a function abstraction

For example, $\lambda f\,((f\,\texttt{true}) :: (f\,\texttt{nil}))$ and $\lambda f\,(f\,f)$ are not typeable in the Mini-ML type system.

Syntactically, because in rule

$$(\mathbf{fn})\,\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x\,(M) : \tau_1 \to \tau_2}$$

the abstracted variable has to be assigned a *trivial* type scheme (recall $x : \tau_1$ stands for $x : \forall\{\,\}\,(\tau_1)$).

Semantically, because $\forall A\,(\tau_1) \to \tau_2$ is not semantically equivalent to an ML type when $A \neq \{\,\}$.

Monomorphic types . . .

$$\tau ::= \alpha \mid bool \mid \tau \to \tau \mid \tau\ list$$

. . . and *type schemes*

$$\sigma ::= \tau \mid \forall \alpha\,(\sigma)$$

Monomorphic types . . .

$$\tau ::= \alpha \mid bool \mid \tau \to \tau \mid \tau\, list$$

. . . and *type schemes*

$$\sigma ::= \tau \mid \forall\alpha\,(\sigma)$$

Polymorphic types

$$\pi ::= \alpha \mid bool \mid \pi \to \pi \mid \pi\, list \mid \forall\alpha\,(\pi)$$

Monomorphic types . . .

$$\tau ::= \alpha \mid \mathit{bool} \mid \tau \to \tau \mid \tau\, \mathit{list}$$

. . . and *type schemes*

$$\sigma ::= \tau \mid \forall \alpha\, (\sigma)$$

Polymorphic types

$$\pi ::= \alpha \mid \mathit{bool} \mid \pi \to \pi \mid \pi\, \mathit{list} \mid \forall \alpha\, (\pi)$$

E.g. $\alpha \to \alpha'$ is a type, $\forall \alpha\, (\alpha \to \alpha')$ is a type scheme and a polymorphic type (but not a monomorphic type), $\forall \alpha\, (\alpha) \to \alpha'$ is a polymorphic type, but not a type scheme.

Identity, Generalisation and Specialisation

$$\textbf{(gen)}\ \frac{\Gamma \vdash M : \pi}{\Gamma \vdash M : \forall \alpha\,(\pi)} \text{ if } \alpha \notin \mathit{ftv}(\Gamma)$$

$$\textbf{(spec)}\ \frac{\Gamma \vdash M : \forall \alpha\,(\pi)}{\Gamma \vdash M : \pi[\pi'/\alpha]}$$

Identity, Generalisation and Specialisation

$$(\mathbf{id})\,\frac{}{\Gamma \vdash x : \pi} \text{ if } (x : \pi) \in \Gamma$$

$$(\mathbf{gen})\,\frac{\Gamma \vdash M : \pi}{\Gamma \vdash M : \forall \alpha\,(\pi)} \text{ if } \alpha \notin \mathit{ftv}(\Gamma)$$

$$(\mathbf{spec})\,\frac{\Gamma \vdash M : \forall \alpha\,(\pi)}{\Gamma \vdash M : \pi[\pi'/\alpha]}$$

[Example 7, p35]

$$\text{(id)}\ \frac{}{f : \forall\alpha(\alpha) \vdash f : \forall\alpha(\alpha)}$$

$$\text{(id)}\ \frac{}{f : \forall\alpha(\alpha) \vdash f : \forall\alpha(\alpha)}$$

[Example 7, p35]

$$(\mathrm{spec})\,\dfrac{(\mathrm{id})\,\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f:\alpha\to\alpha}$$

$$(\mathrm{Spec})\,\dfrac{(\mathrm{id})\,\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f:\alpha}$$

[Example 7, p35]

$$\text{(app)}\,\dfrac{\text{(spec)}\,\dfrac{\text{(id)}\,\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f:\alpha\to\alpha} \qquad \text{(spec)}\,\dfrac{\text{(id)}\,\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f:\alpha}}{f:\forall\alpha(\alpha)\vdash ff:\alpha}$$

[Example 7, p35]

$$
\text{(fn)}\,\dfrac{\text{(gen)}\,\dfrac{\text{(app)}\,\dfrac{\text{(spec)}\,\dfrac{\text{(id)}\,\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f:\alpha\to\alpha}\qquad \text{(spec)}\,\dfrac{\text{(id)}\,\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f:\alpha}}{f:\forall\alpha(\alpha)\vdash ff:\alpha}}{f:\forall\alpha(\alpha)\vdash ff:\forall\alpha(\alpha)}}{\{\}\vdash\lambda f(ff):\forall\alpha(\alpha)\to\forall\alpha(\alpha)}
$$

ML + full polymorphic types has undecidable type-checking

Fact (Wells, 1994). For the modified Mini-ML type system with

- full polymorphic types replacing types and type schemes
- $(\mathbf{id}) + (\mathbf{gen}) + (\mathbf{spec})$ replacing $(\mathbf{var} \succ)$

the type checking and typeability problems are undecidable.

Explicitly versus implicitly typed languages

Implicit: little or no type information is included in program phrases and typings have to be inferred, ideally, entirely at compile-time. (E.g. Standard ML.)

Explicitly versus implicitly typed languages

Implicit: little or no type information is included in program phrases and typings have to be inferred, ideally, entirely at compile-time. (E.g. Standard ML.)

Explicit: most, if not all, types for phrases are explicitly part of the syntax. (E.g. Java.)

Explicitly versus implicitly typed languages

Implicit: little or no type information is included in program phrases and typings have to be inferred, ideally, entirely at compile-time. (E.g. Standard ML.)

Explicit: most, if not all, types for phrases are explicitly part of the syntax. (E.g. Java.)

E.g. self application function of type $\forall\alpha\,(\alpha) \to \forall\alpha\,(\alpha)$
(cf. Example 7)
Implicitly typed version: $\lambda f\,(f\,f)$
Explicitly type version: $\lambda f : \forall\alpha_1\,(\alpha_1)\,(\Lambda\alpha_2\,(f(\alpha_2 \to \alpha_2)(f\,\alpha_2)))$ in PLC...

PLC syntax

Polymorphic
Lambda
Calculus

Types

$$\begin{array}{lcll} \tau & ::= & \alpha & \text{type variable} \\ & | & \tau \to \tau & \text{function type} \\ & | & \forall \alpha\,(\tau) & \forall\text{-type} \end{array}$$

PLC syntax

Types

τ	$::=$	α	type variable
	$\mid$	$\tau \to \tau$	function type
	$\mid$	$\forall \alpha\,(\tau)$	$\forall$-type

Expressions

M	$::=$	x	variable
	$\mid$	$\lambda x : \tau\,(M)$	function abstraction
	$\mid$	$M\,M$	function application
	$\mid$	$\Lambda \alpha\,(M)$	type generalisation
	$\mid$	$M\,\tau$	type specialisation

PLC syntax

Types

$$\begin{array}{lcll} \tau & ::= & \alpha & \text{type variable} \\ & | & \tau \to \tau & \text{function type} \\ & | & \forall \alpha\,(\tau) & \forall\text{-type} \end{array}$$

Expressions

$$\begin{array}{lcll} M & ::= & x & \text{variable} \\ & | & \lambda x : \tau\,(M) & \text{function abstraction} \\ & | & M\,M & \text{function application} \\ & | & \Lambda \alpha\,(M) & \text{type generalisation} \\ & | & M\,\tau & \text{type specialisation} \end{array}$$

(α and x range over fixed, countably infinite sets $\mathbf{TyVar}$ and $\mathbf{Var}$ respectively.)

PLC typing judgement

takes the form $\boxed{\Gamma \vdash M : \tau}$ where

- the *typing environment* Γ is a finite function from variables to PLC types.
  (We write $\Gamma = \{x_1 : \tau_1, \dots, x_n : \tau_n\}$ to indicate that Γ has domain of definition $dom(\Gamma) = \{x_1, \dots, x_n\}$ and maps each x_i to the PLC type τ_i for $i = 1 \dots n$.)
- M is a PLC expression
- τ is a PLC type.

PLC type system

$$(\mathbf{var})\ \frac{}{\Gamma \vdash x : \tau}\ \text{if } (x : \tau) \in \Gamma$$

PLC type system

$$(\mathbf{var})\ \frac{}{\Gamma \vdash x : \tau}\ \text{if } (x : \tau) \in \Gamma$$

$$(\mathbf{fn})\ \frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x : \tau_1\,(M) : \tau_1 \to \tau_2}\ \text{if } x \notin dom(\Gamma)$$

$$(\mathbf{app})\ \frac{\Gamma \vdash M : \tau_1 \to \tau_2 \qquad \Gamma \vdash M' : \tau_1}{\Gamma \vdash M\,M' : \tau_2}$$

PLC type system

$$(\mathbf{var})\;\frac{}{\Gamma \vdash x : \tau}\;\text{if } (x : \tau) \in \Gamma$$

$$(\mathbf{fn})\;\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x : \tau_1\,(M) : \tau_1 \to \tau_2}\;\text{if } x \notin \mathit{dom}(\Gamma)$$

$$(\mathbf{app})\;\frac{\Gamma \vdash M : \tau_1 \to \tau_2 \qquad \Gamma \vdash M' : \tau_1}{\Gamma \vdash M\,M' : \tau_2}$$

$$(\mathbf{gen})\;\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \Lambda\alpha\,(M) : \forall\alpha\,(\tau)}\;\text{if } \alpha \notin \mathit{ftv}(\Gamma)$$

PLC type system

$$(\mathbf{var})\,\frac{}{\Gamma \vdash x : \tau}\quad \text{if } (x : \tau) \in \Gamma$$

$$(\mathbf{fn})\,\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x : \tau_1\,(M) : \tau_1 \to \tau_2}\quad \text{if } x \notin \mathit{dom}(\Gamma)$$

$$(\mathbf{app})\,\frac{\Gamma \vdash M : \tau_1 \to \tau_2 \qquad \Gamma \vdash M' : \tau_1}{\Gamma \vdash M\,M' : \tau_2}$$

$$(\mathbf{gen})\,\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \Lambda\alpha\,(M) : \forall\alpha\,(\tau)}\quad \text{if } \alpha \notin \mathit{ftv}(\Gamma)$$

$$(\mathbf{spec})\,\frac{\Gamma \vdash M : \forall\alpha\,(\tau_1)}{\Gamma \vdash M\,\tau_2 : \tau_1[\tau_2/\alpha]}$$

[Example 12, p 41]

$$(\mathrm{var})\,\frac{}{f : \forall\alpha(\alpha) \vdash f : \forall\alpha(\alpha)} \qquad (\mathrm{var})\,\frac{}{f : \forall\alpha(\alpha) \vdash f : \forall\alpha(\alpha)}$$

[Example 12, p 41]

$$(\text{var})\,\frac{}{f : \forall\alpha(\alpha) \vdash f : \forall\alpha(\alpha)}$$

$$(\text{spec})\,\frac{}{f : \forall\alpha(\alpha) \vdash f\,(\alpha\to\alpha) : \alpha\to\alpha}$$

$$(\text{var})\,\frac{}{f : \forall\alpha(\alpha) \vdash f : \forall\alpha(\alpha)}$$

$$(\text{spec})\,\frac{}{f : \forall\alpha(\alpha) \vdash f\alpha : \alpha}$$

[Example 12, p 41]

$$\text{(app)}\,\dfrac{\text{(spec)}\,\dfrac{\text{(var)}\,\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f(\alpha\to\alpha):\alpha\to\alpha}\qquad \text{(spec)}\,\dfrac{\text{(var)}\,\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f\alpha:\alpha}}{f:\forall\alpha(\alpha)\vdash f(\alpha\to\alpha)(f\alpha):\alpha}$$

[Example 12, p 41]

$$(\text{gen})\dfrac{(\text{app})\dfrac{(\text{spec})\dfrac{(\text{var})\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f(\alpha\to\alpha):\alpha\to\alpha} \qquad (\text{spec})\dfrac{(\text{var})\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f\alpha:\alpha}}{f:\forall\alpha(\alpha)\vdash f(\alpha\to\alpha)(f\alpha):\alpha}}{f:\forall\alpha(\alpha)\vdash \Lambda\alpha(f(\alpha\to\alpha)(f\alpha)):\forall\alpha(\alpha)}$$

[Example 12, p 41]

$$\text{(fn)}\dfrac{\text{(gen)}\dfrac{\text{(app)}\dfrac{\text{(spec)}\dfrac{\text{(var)}\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f(\alpha\to\alpha):\alpha\to\alpha}\qquad \text{(spec)}\dfrac{\text{(var)}\dfrac{}{f:\forall\alpha(\alpha)\vdash f:\forall\alpha(\alpha)}}{f:\forall\alpha(\alpha)\vdash f\alpha:\alpha}}{f:\forall\alpha(\alpha)\vdash f(\alpha\to\alpha)(f\alpha):\alpha}}{f:\forall\alpha(\alpha)\vdash \Lambda\alpha(f(\alpha\to\alpha)(f\alpha)):\forall\alpha(\alpha)}}{\{\}\vdash \lambda f:\forall\alpha(\alpha)(\Lambda\alpha(f(\alpha\to\alpha)(f\alpha))):\forall\alpha(\alpha)\to\forall\alpha(\alpha)}$$

PLC binding forms

$$\forall\alpha(-) \qquad \lambda x : \tau(-) \qquad \Lambda\alpha(-)$$

Eg.

$$\lambda x : \forall\alpha(\beta)\Big(\Lambda\alpha\big(x(\alpha\to\beta)\big)\Big)$$

PLC binding forms

$$\forall\alpha(-) \qquad \lambda x:\tau(-\;) \qquad \Lambda\alpha(\quad)$$

Eg.

$$\lambda x : \forall\beta(\underset{\uparrow\atop\text{free}}{\alpha})\,(\Lambda\alpha(x(\alpha\to\underset{\uparrow\atop\text{free}}{\beta})))$$

An incorrect proof

$$\textbf{(wrong!)}\;\dfrac{\textbf{(fn)}\;\dfrac{\textbf{(var)}\;\dfrac{}{x_1 : \alpha, x_2 : \alpha \vdash x_2 : \alpha}}{x_1 : \alpha \vdash \lambda x_2 : \alpha\,(x_2) : \alpha \to \alpha}}{x_1 : \alpha \vdash \Lambda\alpha\,(\lambda x_2 : \alpha\,(x_2)) : \forall\alpha\,(\alpha \to \alpha)}$$

cos'

$\alpha \in ftv\{x_1 : \alpha\}$

A~~n~~ ~~in~~correct proof

$$(\overset{\text{gen}}{\cancel{\text{wrong}}}\text{!})\ \dfrac{(\text{fn})\ \dfrac{(\text{var})\ \dfrac{}{x_1 : \alpha, x_2 : \alpha' \vdash x_2 : \alpha'}}{x_1 : \alpha \vdash \lambda x_2 : \alpha'(x_2) : \alpha' \to \alpha'}}{x_1 : \alpha \vdash \underbrace{\Lambda\alpha'(\lambda x_2 : \alpha'(x_2))}_{\begin{array}{c}\shortparallel \\ \Lambda\alpha(\lambda x_2 : \alpha(x_2))\end{array}} : \underbrace{\forall\alpha'(\alpha' \to \alpha')}_{\begin{array}{c}\shortparallel \\ \forall\alpha(\alpha \to \alpha)\end{array}}}$$

Decidability of the PLC typeability and type-checking problems

Theorem.
For each PLC typing problem, $\Gamma \vdash M : ?$, there is at most one PLC type τ for which $\Gamma \vdash M : \tau$ is provable. Moreover there is an algorithm, typ, which when given any $\Gamma \vdash M : ?$ as input, returns such a τ if it exists and FAILs otherwise.

(N.B. equality of PLC types up to alpha-conversion is decidable.)

Decidability of the PLC typeability and type-checking problems

Theorem.
For each PLC typing problem, $\Gamma \vdash M : ?$, there is at most one PLC type τ for which $\Gamma \vdash M : \tau$ is provable. Moreover there is an algorithm, typ, which when given any $\Gamma \vdash M : ?$ as input, returns such a τ if it exists and $FAIL$s otherwise.

Corollary.
The PLC type checking problem is decidable: we can decide whether or not $\Gamma \vdash M : \tau$ is provable by checking whether $typ(\Gamma \vdash M : ?) = \tau$.

(N.B. equality of PLC types up to alpha-conversion is decidable.)

PLC type-checking algorithm, I

Variables

$typ(\Gamma, x : \tau \vdash x : ?) \triangleq \tau$

PLC type-checking algorithm, I

Variables

$$typ(\Gamma, x : \tau \vdash x : ?) \triangleq \tau$$

Function abstractions

$$typ(\Gamma \vdash \lambda x : \tau_1\,(M) : ?) \triangleq$$
$$\textbf{let}\ \ \tau_2 = typ(\Gamma, x : \tau_1 \vdash M : ?)\ \textbf{in}\ \tau_1 \to \tau_2$$

PLC type-checking algorithm, I

Variables

$$\mathit{typ}(\Gamma, x : \tau \vdash x : ?) \triangleq \tau$$

Function abstractions

$$\mathit{typ}(\Gamma \vdash \lambda x : \tau_1\,(M) : ?) \triangleq$$
$$\textbf{let}\ \ \tau_2 = \mathit{typ}(\Gamma, x : \tau_1 \vdash M : ?)\ \textbf{in}\ \tau_1 \to \tau_2$$

Function applications

$$\begin{array}{l}
\mathit{typ}(\Gamma \vdash M_1\, M_2 : ?) \triangleq \\
\ \textbf{let}\ \tau_1 = \mathit{typ}(\Gamma \vdash M_1 : ?)\ \textbf{in} \\
\ \textbf{let}\ \tau_2 = \mathit{typ}(\Gamma \vdash M_2 : ?)\ \textbf{in} \\
\ \textbf{case}\ \tau_1\ \textbf{of}\ \ \tau \to \tau' \ \mapsto\ \textbf{if}\ \tau = \tau_2\ \textbf{then}\ \tau'\ \textbf{else}\ \mathit{FAIL} \\
\qquad\qquad\ \ \mid\qquad \_ \quad\ \mapsto\ \mathit{FAIL}
\end{array}$$

PLC type-checking algorithm, II

Type generalisations

$\boldsymbol{typ(\Gamma \vdash \Lambda\alpha\ (M) : ?)} \triangleq$
$\quad \textbf{let } \boldsymbol{\tau = typ(\Gamma \vdash M : ?)} \textbf{ in } \boldsymbol{\forall\alpha\ (\tau)}$

PLC type-checking algorithm, II

Type generalisations

$$
\begin{array}{l}
\mathit{typ}(\Gamma \vdash \Lambda\alpha\,(M) : ?) \triangleq \\
\ \ \mathbf{let}\ \tau = \mathit{typ}(\Gamma \vdash M : ?)\ \mathbf{in}\ \forall\alpha\,(\tau)
\end{array}
$$

Type specialisations

$$
\begin{array}{l}
\mathit{typ}(\Gamma \vdash M\,\tau_2 : ?) \triangleq \\
\ \ \mathbf{let}\ \tau = \mathit{typ}(\Gamma \vdash M : ?)\ \mathbf{in} \\
\ \ \mathbf{case}\ \tau\ \mathbf{of}\ \ \forall\alpha\,(\tau_1)\ \mapsto\ \tau_1[\tau_2/\alpha] \\
\qquad\qquad\ \ \ \mid\qquad\quad \_\ \ \mapsto\ \mathit{FAIL}
\end{array}
$$