Mini-ML type-inference problem:
given $\Gamma$ & $\Delta$, does there exist $\sigma$ such that $\Gamma \vdash \Delta : \sigma$ holds?
Two examples involving self-application

\[ M \triangleq \text{let } f = \lambda x_1 (\lambda x_2 (x_1)) \text{ in } f \ f \]

\[ M' \triangleq (\lambda f (f \ f)) \lambda x_1 (\lambda x_2 (x_1)) \]

Are \( M \) and \( M' \) typeable in the Mini-ML type system?

(in the empty typing environment)
\[
\begin{align*}
\text{(C3)} & \quad x_1 : x_2 : \vdash x_1 : \\
\text{(C2)} & \quad x_1 : \vdash \lambda x_2(x_1) : \\
\text{(C1)} & \quad \{\} \vdash \lambda x_1 (\lambda x_2(x_1)) : \\
\text{(C5)} & \quad f : \vdash \text{let } f = \lambda x_1 (\lambda x_2(x_1)) \text{ in } f f : \\
\text{(C6)} & \quad \vdash f : f : \\
\text{(C4)} & \quad f : \vdash \text{let } f f : \\
\text{(C0)} & \quad \{\} \vdash \text{let } f = \lambda x_1 (\lambda x_2(x_1)) \text{ in } f f : 
\end{align*}
\]
\[ x_1 : \tau_3, \ x_2 : \tau_5 \vdash x_1 : \tau_6 \]  
\[ x_1 : \tau_3 \vdash \lambda x_2(x_1) : \tau_4 \]  
\[ \{\} \vdash \lambda x_1(\lambda x_2(x_1)) : \tau_2 \]  
\[ \{\} \vdash \text{let } f = \lambda x_1(\lambda x_2(x_1)) \text{ in } f f : \tau_1 \]
Constraints generated while inferring a type for
\[
\text{let } f = \lambda x_1 (\lambda x_2 (x_1)) \text{ in } f f
\]

\[
A = ftv(\tau_2) \quad (C0)
\]
\[
\tau_2 = \tau_3 \rightarrow \tau_4 \quad (C1)
\]
\[
\tau_4 = \tau_5 \rightarrow \tau_6 \quad (C2)
\]
\[
\forall \{ \} (\tau_3) \succ \tau_6, \text{ i.e. } \tau_3 = \tau_6 \quad (C3)
\]
\[
\tau_7 = \tau_8 \rightarrow \tau_1 \quad (C4)
\]
\[
\forall A (\tau_2) \succ \tau_7 \quad (C5)
\]
\[
\forall A (\tau_2) \succ \tau_8 \quad (C6)
\]
\[ \tau_2 \equiv \tau_3 \rightarrow \tau_4 \equiv \tau_3 \rightarrow (\tau_5 \rightarrow \tau_6) \equiv \tau_6 \rightarrow (\tau_5 \rightarrow \tau_6) \]
\[\tau_2 \Rightarrow \tau_3 \Rightarrow \tau_4 \Rightarrow \tau_3 \Rightarrow (\tau_5 \Rightarrow \tau_6) \Rightarrow \tau_6 \Rightarrow (\tau_5 \Rightarrow \tau_6)\]

Take \[\tau_6 = \alpha_1\] \{ type variables. \]

So \[A = \text{ftv}(\tau_2) = \text{ftv}(\alpha_1 \Rightarrow (\alpha_2 \Rightarrow \alpha_1)) = \{\alpha_1, \alpha_2\}\]
\[
\begin{align*}
\tau_2 & \xRightarrow{(C1)} \tau_3 \to \tau_4 \xRightarrow{(C2)} \tau_3 \to (\tau_5 \to \tau_6) \xRightarrow{(C3)} \tau_6 \to (\tau_5 \to \tau_6) \\
\text{Take } & \quad \tau_6 = \alpha_1 \} \text{ type variables.} \\
\tau_5 & = \alpha_2
\end{align*}
\]

So \( A = \text{ftv}(\tau_2) = \text{ftv}(\alpha_1 \to (\alpha_2 \to \alpha_1)) = \{ \alpha_1, \alpha_2 \} \)

\((C5)\) : \( \forall \alpha_1, \alpha_2 \) \( (\alpha_1 \to (\alpha_2 \to \alpha_1)) \not\supset \tau_7 \xRightarrow{(C4)} \tau_8 \to \tau_1 \)

\((C6)\) : \( \quad \quad \quad \quad \quad \not\supset \tau_8 \)

So \( \{ \tau_8 \to \tau_1 = \tau_9 \to (\tau_{10} \to \tau_9) \} \) for some \( \tau_9, \tau_{10}, \tau_{11}, \tau_{12} \)
\[
\begin{align*}
\tau_2 \overset{(C1)}{=} \tau_3 \rightarrow \tau_4 \overset{(C2)}{=} \tau_3 \rightarrow (\tau_5 \rightarrow \tau_6) \overset{(C3)}{=} \tau_6 \rightarrow (\tau_5 \rightarrow \tau_6)
\end{align*}
\]

Take \[
\begin{align*}
\tau_6 & = \alpha_1 \\
\tau_5 & = \alpha_2
\end{align*}
\]

So \[
A = \text{ftv}(\tau_2) = \text{ftv}(\alpha_1 \rightarrow (\alpha_2 \rightarrow \alpha_1)) = \{ \alpha_1, \alpha_2 \}
\]

(C5): \[
\forall \alpha_1, \alpha_2 \forall (\alpha_1 \rightarrow (\alpha_2 \rightarrow \alpha_1)) > \tau_7 \overset{(C4)}{=} \tau_8 \rightarrow \tau_1
\]

(C6): \[
\text{" } \tau_7 \text{" } \rightarrow \tau_8
\]

So \[
\begin{align*}
\& \{ \tau_8 = \tau_9 \land \tau_1 = (\tau_{10} \rightarrow \tau_9) \text{ for some } \tau_9, \tau_{10}, \\
& \tau_8 = \tau_{11} \rightarrow (\tau_{12} \rightarrow \tau_{11}) \}
\end{align*}
\]

So \[
\tau_1 = \tau_{10} \rightarrow \tau_9 = \tau_{10} \rightarrow \tau_8 = \tau_{10} \rightarrow (\tau_{11} \rightarrow (\tau_{12} \rightarrow \tau_{11}))
\]
Thus

\{ \{ \} \} \vdash (\text{let } f = \lambda x_1 (\lambda x_2 (x_1)) \text{ in } \text{ff} ) : \tau_\_0 \rightarrow (\tau_\_1 \rightarrow (\tau_\_2 \rightarrow \tau_\_1))

holds for any \( \tau_\_0, \tau_\_1, \tau_\_2 \)

So

\vdash (\text{let } f = \lambda x_1 (\lambda x_2 (x_1)) \text{ in } \text{ff} ) :

\forall \alpha_1, \alpha_2, \alpha_3 (\alpha_1 \rightarrow (\alpha_2 \rightarrow (\alpha_3 \rightarrow \alpha_2)))
Two examples involving self-application

$M \triangleq \text{let } f = \lambda x_1 (\lambda x_2 (x_1)) \text{ in } f \ f$

$M' \triangleq (\lambda f (f \ f)) \lambda x_1 (\lambda x_2 (x_1))$

Are $M$ and $M'$ typeable in the Mini-ML type system?
The constraints generated from trying to type

\[(\lambda f(\lambda f f)) \lambda x_1(\lambda x_2(x_1))\]

give

\[\tau_7 \equiv \tau_4 \equiv \tau_6 \equiv \tau_7 \rightarrow \tau_5\]
The constraints generated from trying to type

\((\lambda f (ff)) \; \lambda x_1 (\lambda x_2 (x_1))\)

give

\[ \tau_7 \overset{(C13)}{=} \tau_4 = \tau_6 = \tau_7 \rightarrow \tau_5 \]

these cannot be equal — they have different numbers of the symbol “→” in them
Two examples involving self-application

\[ M \triangleq \text{let } f = \lambda x_1 (\lambda x_2 (x_1)) \text{ in } f f \]

\[ M' \triangleq (\lambda f (f f)) \lambda x_1 (\lambda x_2 (x_1)) \]

Are \( M \) and \( M' \) typeable in the Mini-ML type system?

This is not typeable
Principal type schemes for closed expressions

A type scheme $\forall A (\tau)$ is the principal type scheme of a closed Mini-ML expression $M$ if
Principal type schemes for closed expressions

A type scheme \( \forall A (\tau) \) is the principal type scheme of a closed Mini-ML expression \( M \) if

\[(a) \vdash M : \forall A (\tau)\]
Principal type schemes for closed expressions

A type scheme $\forall A (\tau)$ is the *principal* type scheme of a closed Mini-ML expression $M$ if

(a) $\vdash M : \forall A (\tau)$

(b) for any other type scheme $\forall A' (\tau')$, if $\vdash M : \forall A' (\tau')$, then $\forall A (\tau) \not\succ \tau'$
Principal type schemes for closed expressions

A type scheme $\forall A \ (\tau)$ is the principal type scheme of a closed Mini-ML expression $M$ if

(a) $\vdash M : \forall A \ (\tau)$

(b) for any other type scheme $\forall A' \ (\tau')$, if $\vdash M : \forall A' \ (\tau')$, then $\forall A \ (\tau) \gg \tau'$

eg $\forall \alpha_1, \alpha_2, \alpha_3 \ (\alpha_1 \to (\alpha_2 \to (\alpha_3 \to \alpha_2)))$ is principal type scheme

let $f = \lambda x_1 (\lambda x_2 (x_1))$ in $ff$
Theorem (Hindley; Damas-Milner)

**Theorem.** If the closed Mini-ML expression $M$ is typeable (i.e. $\vdash M : \sigma$ holds for some type scheme $\sigma$), then there is a principal type scheme for $M$. 
Theorem. If the closed Mini-ML expression $M$ is typeable (i.e. $\vdash M : \sigma$ holds for some type scheme $\sigma$), then there is a principal type scheme for $M$.

Indeed, there is an algorithm which, given any closed Mini-ML expression $M$ as input, decides whether or not it is typeable and returns a principal type scheme if it is.
An ML expression with a principal type scheme hundreds of pages long

```
let pair = \x (\y (\z (x y))) in
let x_1 = \y (pair y y) in
    let x_2 = \y (x_1 (x_1 y)) in
        let x_3 = \y (x_2 (x_2 y)) in
            let x_4 = \y (x_3 (x_3 y)) in
                let x_5 = \y (x_4 (x_4 y)) in
                    x_5 (\y y)
```
Unification of ML types

There is an algorithm \textit{mgu} which when input two Mini-ML types $\tau_1$ and $\tau_2$ decides whether $\tau_1$ and $\tau_2$ are \textit{unifiable}, i.e. whether there exists a type-substitution $S \in \text{Sub}$ with
Unification of ML types

There is an algorithm \textit{mgu} which when input two Mini-ML types \(\tau_1\) and \(\tau_2\) decides whether \(\tau_1\) and \(\tau_2\) are \textit{unifiable}, i.e. whether there exists a type-substitution \(S \in \text{Sub}\) with

(a) \(S(\tau_1) = S(\tau_2)\).
Unification of ML types

There is an algorithm \texttt{mgu} which when input two Mini-ML types \(\tau_1\) and \(\tau_2\) decides whether \(\tau_1\) and \(\tau_2\) are \textit{unifiable}, i.e. whether there exists a type-substitution \(S \in \text{Sub}\) with

\[(a)\ S(\tau_1) = S(\tau_2).\]

Moreover, if they are unifiable, \(\texttt{mgu}(\tau_1, \tau_2)\) returns the \textit{most general unifier}—an \(S\) satisfying both \((a)\) and
Unification of ML types

There is an algorithm \textit{mgu} which when input two Mini-ML types \( \tau_1 \) and \( \tau_2 \) decides whether \( \tau_1 \) and \( \tau_2 \) are \textit{unifiable}, i.e. whether there exists a type-substitution \( S \in \text{Sub} \) with

(a) \( S(\tau_1) = S(\tau_2) \).

Moreover, if they are unifiable, \( \text{mgu}(\tau_1, \tau_2) \) returns the \textit{most general unifier}—an \( S \) satisfying both (a) and

(b) for all \( S' \in \text{Sub} \), if \( S'(\tau_1) = S'(\tau_2) \), then \( S' = TS \) for some \( T \in \text{Sub} \).
Unification of ML types

There is an algorithm $\text{mgu}$ which when input two Mini-ML types $\tau_1$ and $\tau_2$ decides whether $\tau_1$ and $\tau_2$ are unifiable, i.e. whether there exists a type-substitution $S \in \text{Sub}$ with

(a) $S(\tau_1) = S(\tau_2)$.

Moreover, if they are unifiable, $\text{mgu}(\tau_1, \tau_2)$ returns the most general unifier—an $S$ satisfying both (a) and

(b) for all $S' \in \text{Sub}$, if $S'(\tau_1) = S'(\tau_2)$, then $S' = TS$ for some $T \in \text{Sub}$

(any other substitution $S'$ can be factored through $S$, by specialising $S$ with $T$)
Unification of ML types

There is an algorithm \textit{mgu} which when input two Mini-ML types \(\tau_1\) and \(\tau_2\) decides whether \(\tau_1\) and \(\tau_2\) are \textit{unifiable}, i.e. whether there exists a type-substitution \(S \in \text{Sub}\) with

\[(a) \ S(\tau_1) = S(\tau_2) .\]

Moreover, if they are unifiable, \(\text{mgu}(\tau_1, \tau_2)\) returns the \textit{most general unifier}—an \(S\) satisfying both (a) and

\[(b) \text{ for all } S' \in \text{Sub}, \text{ if } S'(\tau_1) = S'(\tau_2), \text{ then } S' = TS \text{ for some } T \in \text{Sub} .\]

(any other substitution \(S'\) can be factored through \(S\), by specialising \(S\) with \(T\))

By convention \(\text{mgu}(\tau_1, \tau_2) = \text{FAIL}\) if (and only if) \(\tau_1\) and \(\tau_2\) are not unifiable.
Principal type schemes for open expressions

A solution for the typing problem $\Gamma \vdash M : ?$ is a pair $\langle S, \sigma \rangle$ consisting of a type substitution $S$ and a type scheme $\sigma$ satisfying
Principal type schemes for open expressions

A *solution* for the typing problem $\Gamma \vdash M : ?$ is a pair $(S, \sigma)$ consisting of a type substitution $S$ and a type scheme $\sigma$ satisfying

$$S \Gamma \vdash M : \sigma$$

(where $S \Gamma = \{x_1 : S \sigma_1, \ldots, x_n : S \sigma_n\}$, if $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$).
Principal type schemes for open expressions

A **solution** for the typing problem $\Gamma \vdash M : ?$ is a pair $(S, \sigma)$ consisting of a type substitution $S$ and a type scheme $\sigma$ satisfying

$$S \Gamma \vdash M : \sigma$$

(where $S \Gamma = \{x_1 : S \sigma_1, \ldots, x_n : S \sigma_n\}$, if $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$).

Such a solution is **principal** if given any other, $(S', \sigma')$, there is some $T \in \text{Sub}$ with $TS = S'$ and $T(\sigma) \vdash \sigma'$.
Principal type schemes for open expressions

A solution for the typing problem $\Gamma \vdash M : ?$ is a pair $(S, \sigma)$ consisting of a type substitution $S$ and a type scheme $\sigma$ satisfying

$$S \Gamma \vdash M : \sigma$$

(where $S \Gamma = \{x_1 : S \sigma_1, \ldots, x_n : S \sigma_n\}$, if $\Gamma = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$).

Such a solution is principal if given any other, $(S', \sigma')$, there is some $T \in \text{Sub}$ with $TS = S'$ and $T(\sigma) \not\geq \sigma'$.

(For type schemes $\sigma$ and $\sigma'$, with $\sigma' = \forall A' (\tau')$ say, we define $\sigma \succ \sigma'$ to mean $A' \cap \text{ftv}(\sigma) = \{\}$ and $\sigma \succ \tau'$.)
Example typing problem and solutions

Typing problem

\[
x : \forall \alpha (\beta \rightarrow (\gamma \rightarrow \alpha)) \vdash x \text{true} : ?
\]

has solutions:

\[
I S_1 = \{b \in \text{bool} \}, s_1 = 8 a (g a)
\]

\[
I S_2 = \{b \in \text{bool}, g \in \text{bool} \}, s_2 = 8 a 0 (a a)
\]

\[
I S_3 = \{b \in \text{bool}, g \in \text{bool} \}, s_3 = 8 \{\} (a a)
\]

\[
I S_4 = \{b \in \text{bool}, g \in \text{bool} \}, s_3 = 8 \{\} (a a)
\]

Both \((S_1, s_1)\) and \((S_2, s_2)\) are in fact principal solutions.
Example typing problem and solutions

Typing problem

\[ x : \forall \alpha (\beta \rightarrow (\gamma \rightarrow \alpha)) \vdash x \text{true} : ? \]

has solutions:

- \( S_1 = \{ \beta \mapsto \text{bool} \} \), \( \sigma_1 = \forall \alpha (\gamma \rightarrow \alpha) \)
Example typing problem and solutions

Typing problem

$$x : \forall \alpha (\beta \rightarrow (\gamma \rightarrow \alpha)) \vdash x \text{true} : ?$$

has solutions:

- $S_1 = \{\beta \mapsto \text{bool}\}, \sigma_1 = \forall \alpha (\gamma \rightarrow \alpha)$

- $S_2 = \{\beta \mapsto \text{bool}, \gamma \mapsto \alpha\}, \sigma_2 = \forall \alpha' (\alpha \rightarrow \alpha')$
Example typing problem and solutions

Typing problem

\[ x : \forall \alpha (\beta \rightarrow (\gamma \rightarrow \alpha)) \vdash x \text{true} : ? \]

has solutions:

- \( S_1 = \{ \beta \mapsto \text{bool} \}, \sigma_1 = \forall \alpha (\gamma \rightarrow \alpha) \)
- \( S_2 = \{ \beta \mapsto \text{bool}, \gamma \mapsto \alpha \}, \sigma_2 = \forall \alpha' (\alpha \rightarrow \alpha') \)
- \( S_3 = \{ \beta \mapsto \text{bool}, \gamma \mapsto \alpha \}, \sigma_3 = \forall \alpha' (\alpha \rightarrow (\alpha' \rightarrow \alpha')) \)

Both \((S_1, s_1)\) and \((S_2, s_2)\) are in fact principal solutions.
Example typing problem and solutions

Typing problem

$$x : \forall \alpha (\beta \to (\gamma \to \alpha)) \vdash x \text{true} : ?$$

has solutions:

- $$S_1 = \{\beta \mapsto \text{bool}\}, \sigma_1 = \forall \alpha (\gamma \to \alpha)$$
- $$S_2 = \{\beta \mapsto \text{bool}, \gamma \mapsto \alpha\}, \sigma_2 = \forall \alpha' (\alpha \to \alpha')$$
- $$S_3 = \{\beta \mapsto \text{bool}, \gamma \mapsto \alpha\}, \sigma_3 = \forall \alpha' (\alpha \to (\alpha' \to \alpha'))$$
- $$S_4 = \{\beta \mapsto \text{bool}, \gamma \mapsto \text{bool}\}, \sigma_3 = \forall \{\} (\text{bool} \to \text{bool})$$
Example typing problem and solutions

Typing problem

\[ x : \forall \alpha (\beta \rightarrow (\gamma \rightarrow \alpha)) \vdash x \text{true} : ? \]

has solutions:

- \( S_1 = \{ \beta \mapsto \text{bool} \}, \sigma_1 = \forall \alpha (\gamma \rightarrow \alpha) \)
- \( S_2 = \{ \beta \mapsto \text{bool}, \gamma \mapsto \alpha \}, \sigma_2 = \forall \alpha' (\alpha \rightarrow \alpha') \)
- \( S_3 = \{ \beta \mapsto \text{bool}, \gamma \mapsto \alpha \}, \sigma_3 = \forall \alpha' (\alpha \rightarrow (\alpha' \rightarrow \alpha')) \)
- \( S_4 = \{ \beta \mapsto \text{bool}, \gamma \mapsto \text{bool} \}, \sigma_3 = \forall \{ \} (\text{bool} \rightarrow \text{bool}) \)

Both \((S_1, \sigma_1)\) and \((S_2, \sigma_2)\) are in fact principal solutions.
Properties of the Mini-ML typing relation with respect to substitution and type scheme specialisation

- If \( \Gamma \vdash M : \sigma \), then for any type substitution \( S \in \text{Sub} \)
  \[ S\Gamma \vdash M : S\sigma \]
Properties of the Mini-ML typing relation with respect to substitution and type scheme specialisation

- If $\Gamma \vdash M : \sigma$, then for any type substitution $S \in \text{Sub}$

$$S\Gamma \vdash M : S\sigma$$

- If $\Gamma \vdash M : \sigma$ and $\sigma \supseteq \sigma'$, then

$$\Gamma \vdash M : \sigma'$$
Requirements for a principal typing algorithm, \( pt \)

\( pt \) operates on typing problems \( \Gamma \vdash M : ? \) (consisting of a typing environment \( \Gamma \) and a Mini-ML expression \( M \)).
Requirements for a principal typing algorithm, $pt$

$pt$ operates on typing problems $\Gamma \vdash M : ?$ (consisting of a typing environment $\Gamma$ and a Mini-ML expression $M$).

It returns either a pair $(S, \tau)$ consisting of a type substitution $S \in \text{Sub}$ and a Mini-ML type $\tau$, or the exception $\text{FAIL}$. 
Requirements for a principal typing algorithm, $pt$

$pt$ operates on typing problems $\Gamma \vdash M : ?$ (consisting of a typing environment $\Gamma$ and a Mini-ML expression $M$).

It returns either a pair $(S, \tau)$ consisting of a type substitution $S \in \text{Sub}$ and a Mini-ML type $\tau$, or the exception $FAIL$.

- If $\Gamma \vdash M : ?$ has a solution (cf. Slide 28), then $pt(\Gamma \vdash M : ?)$ returns $(S, \tau)$ for some $S$ and $\tau$;
Requirements for a principal typing algorithm, \( pt \)

\( pt \) operates on typing problems \( \Gamma \vdash M : ? \) (consisting of a typing environment \( \Gamma \) and a Mini-ML expression \( M \)).

It returns either a pair \( (S, \tau) \) consisting of a type substitution \( S \in \text{Sub} \) and a Mini-ML type \( \tau \), or the exception \text{FAIL}.

- If \( \Gamma \vdash M : ? \) has a solution (cf. Slide 28), then \( pt(\Gamma \vdash M : ?) \) returns \( (S, \tau) \) for some \( S \) and \( \tau \); moreover, setting \( A = (ftv(\tau) - ftv(S \Gamma)) \), then \( (S, \forall A(\tau)) \) is a principal solution for the problem \( \Gamma \vdash M : ? \).
Requirements for a principal typing algorithm, \( pt \)

\( pt \) operates on typing problems \( \Gamma \vdash M : ? \) (consisting of a typing environment \( \Gamma \) and a Mini-ML expression \( M \)).

It returns either a pair \( (S, \tau) \) consisting of a type substitution \( S \in \text{Sub} \) and a Mini-ML type \( \tau \), or the exception \( \text{FAIL} \).

- If \( \Gamma \vdash M : ? \) has a solution (cf. Slide 28), then \( pt(\Gamma \vdash M : ?) \) returns \( (S, \tau) \) for some \( S \) and \( \tau \); moreover, setting \( A = (ftv(\tau) \leftarrow ftv(S \Gamma)) \), then \( (S, \forall A(\tau)) \) is a principal solution for the problem \( \Gamma \vdash M : ? \).

- If \( \Gamma \vdash M : ? \) has no solution, then \( pt(\Gamma \vdash M : ?) \) returns \( \text{FAIL} \).
How the principal typing algorithm $pt$ works

$$pt(\Gamma \vdash M : ?) = (S, \tau) \mid FAIL$$

- Call $pt$ recursively following the structure of $M$ and guided by the typing rules, bottom-up.
- Thread substitutions sequentially and compose them together when returning from a recursive call.
- When types need to agree to satisfy a typing rule, use $mgu$ (and $pt$ returns $FAIL$ only if $mgu$ does).
- When types are unknown, generate a fresh type variable.
Some of the clauses in a definition of \(pt\)

*Function abstractions:* \(pt(\Gamma \vdash \lambda x (M) : ?) \triangleq \)

let \(\alpha = \text{fresh}\) in

let \((S, \tau) = pt(\Gamma, x : \alpha \vdash M : ?)\) in \((S, S(\alpha) \rightarrow \tau)\)
Some of the clauses in a definition of \( pt \)

**Function abstractions:** \( pt(\Gamma \vdash \lambda x (M) : ?) \triangleq \)

\[
\begin{align*}
& \text{let } \alpha = \text{fresh} \text{ in} \\
& \text{let } (S, \tau) = pt(\Gamma, x : \alpha \vdash M : ?) \text{ in } (S, S(\alpha) \rightarrow \tau)
\end{align*}
\]

**Function applications:** \( pt(\Gamma \vdash M_1 \ M_2 : ?) \triangleq \)

\[
\begin{align*}
& \text{let } (S_1, \tau_1) = pt(\Gamma \vdash M_1 : ?) \text{ in} \\
& \text{let } (S_2, \tau_2) = pt(S_1 \ \Gamma \vdash M_2 : ?) \text{ in} \\
& \text{let } \alpha = \text{fresh} \text{ in} \\
& \text{let } S_3 = mgu(S_2 \ \tau_1, \tau_2 \rightarrow \alpha) \text{ in } (S_3 S_2 S_1, S_3(\alpha))
\end{align*}
\]
Mini-ML type system, III

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

(app) \[
\frac{\Gamma \vdash M : \tau_1 \to \tau_2 \quad \Gamma \vdash N : \tau_1}{\Gamma \vdash MN : \tau_2}
\]
\[ \text{pt}(\Gamma \vdash M_1 : ?) = (S_1, \tau_1) \]

\[ S_1, \Gamma \vdash M_1 : \tau_1 \]

\[ \downarrow \text{pt}(\Gamma' \vdash M_1M_2 : ?) = \]
\[
\text{pt}(\Gamma \vdash M_1:?) = (S_1, \tau_1)
\]

\[
\text{pt}(S_1 \Gamma \vdash M_2:?) = (S_2, \tau_2)
\]

\[
\rightarrow \text{pt}(\Gamma' \vdash M_1 M_2:?) = 
\]

\[
S_2 S_1 \Gamma \vdash M_1 : S_2 \tau_1
\]

\[
S_2 S_1 \Gamma' \vdash M_2 : \tau_2
\]

+slide 28
\[\text{pt}(\Gamma \vdash M_1 : ?) = (S_1, \tau_1)\]

\[\text{mgun}(S_2 \tau_1, \tau_2 \rightarrow \alpha) = S_3\]

\[S_3 \tau_2 \rightarrow S_3 \alpha\]

\[S_3 S_2 S_1 \Gamma \vdash M_1 : S_3 S_2 \tau_1\]

\[\text{pt}(S_1 \Gamma \vdash M_2 : ?) = (S_2, \tau_2)\]

\[\text{Slide 28}\]

\[S_3 S_2 S_1 \Gamma' \vdash M_2 : S_3 \tau_2\]

\[\text{pt}(\Gamma' \vdash M_1 M_2 : ?) = (S_3 S_2 S_1, S_3 \alpha)\]
\[ \text{pt}(\Gamma \vdash M_1 : ?) = (S_1, \tau_1) \]
\[ \text{pt}(S_1 \Gamma \vdash M_2 : ?) = (S_2, \tau_2) \]
\[ \text{mg u}(S_2 \tau_1, \tau_2 \rightarrow \alpha) = S_3 \]

\[ S_3 S_2 S_1 \Gamma \vdash M_1 : S_3 \tau_2 \rightarrow S_3 \alpha \]

\[ \begin{array}{c}
S_3 S_2 S_1 \Gamma \vdash M_1 M_2 : S_3 \alpha \\
\downarrow \text{pt} \quad \downarrow \text{pt} \quad \downarrow \text{pt} \\
(S_3 S_2 S_1, S_3 \alpha) \end{array} \]

\[ (\text{app}) \]