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

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

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

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

(in the empty typing environment)
\begin{align*}
\text{(C1)} & \quad \{ \} \vdash \lambda x_1(\lambda x_2(x_1)) : \\
\text{(C2)} & \quad x_1 : \vdash \lambda x_2(x_1) : \\
\text{(C3)} & \quad x_1, x_2 : \vdash x_1 : \\
\text{(C4)} & \quad f : \quad \vdash f : f : \\
\text{(C5)} & \quad f : \quad \vdash f : \\
\text{(C6)} & \quad f : \quad \vdash f : \\
\text{(C0)} & \quad \{ \} \vdash \text{let } f = \lambda x_1(\lambda x_2(x_1)) \text{ in } f f :
\end{align*}
\[
\begin{align*}
\text{(C1)} & \quad \{ \} \vdash \lambda x_{1}(\lambda x_{2}(x_{1})) : \tau_{2} \\
\text{(C2)} & \quad x_{1} : \tau_{3}, x_{2} : \tau_{5} \vdash x_{1} : \tau_{6} \\
\text{(C3)} & \quad x_{1} : \tau_{3} \vdash \lambda x_{2}(x_{1}) : \tau_{4} \\
\text{(C4)} & \quad f : \forall A(\tau_{2}) \vdash f : \tau_{7} \\
\text{(C5)} & \quad f : \forall A(\tau_{2}) \vdash f : \tau_{8} \\
\text{(C6)} & \quad f : \forall A(\tau_{2}) \vdash ff : \tau_{1} \\
\text{(C0)} & \quad \{ \} \vdash \text{let } f = \lambda x_{1}(\lambda x_{2}(x_{1})) \text{ in } ff : \tau_{1}
\end{align*}
\]
Constraints generated while inferring a type for

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

\[ A = ftv(\tau_2) \quad (C0) \]
\[ \tau_2 = \tau_3 \rightarrow \tau_4 \quad (C1) \]
\[ \tau_4 = \tau_5 \rightarrow \tau_6 \quad (C2) \]
\[ \forall \{ \} (\tau_3) \succ \tau_6, \text{ i.e. } \tau_3 = \tau_6 \quad (C3) \]
\[ \tau_7 = \tau_8 \rightarrow \tau_1 \quad (C4) \]
\[ \forall A (\tau_2) \succ \tau_7 \quad (C5) \]
\[ \forall A (\tau_2) \succ \tau_8 \quad (C6) \]
\( T_2 \overset{(C1)}{=} T_3 \rightarrow T_4 \overset{(C2)}{=} T_3 \rightarrow (T_5 \rightarrow T_6) \overset{(C3)}{=} T_6 \rightarrow (T_5 \rightarrow T_6) \)
\[ T_2 \overset{(C1)}{=} T_3 \rightarrow T_4 \overset{(C2)}{=} T_3 \rightarrow (T_5 \rightarrow T_6) \overset{(C3)}{=} T_6 \rightarrow (T_5 \rightarrow T_6) \]

Take \( T_6 = \alpha_1 \) \( \{\} \) type variables.

So \( A = ftv(T_2) = ftv(\alpha_1 \rightarrow (\alpha_2 \rightarrow \alpha_1)) = \{\alpha_1, \alpha_2\} \)
\[ \tau_2 \overset{(C1)}{=} \tau_3 \rightarrow \tau_4 \overset{(C2)}{=} \tau_3 \rightarrow (\tau_5 \rightarrow \tau_6) \overset{(C3)}{=} \tau_6 \rightarrow (\tau_5 \rightarrow \tau_6) \]

Take \[ \tau_6 = \alpha_1 \]
\[ \tau_5 = \alpha_2 \]
\}

Type variables.

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

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

(C6) \[ \tau_8 \rightarrow \tau_1 = \tau_9 \rightarrow (\tau_{10} \rightarrow \tau_9) \]

for some \[ \tau_9, \tau_{10}, \tau_{11}, \tau_{12} \]

So \[ \{ \tau_8 \rightarrow \tau_1 = \tau_9 \rightarrow (\tau_{10} \rightarrow \tau_9) \]
\[ (C1) \quad \tau_2 \Rightarrow \tau_3 \rightarrow \tau_4 \Rightarrow \tau_5 \rightarrow \tau_6 \rightarrow \tau_7 \Rightarrow \tau_8 \rightarrow \tau_1 \]

Take \[ \tau_6 = \alpha_1 \]
\[ \tau_5 = \alpha_2 \] \{ type variables. \}

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

\[ (C5) \quad \forall \alpha_1, \alpha_2 \exists (\alpha_1 \rightarrow (\alpha_2 \rightarrow \alpha_1)) > \tau_7 \Rightarrow \tau_8 \rightarrow \tau_1 \]

\[ (C4) \quad \forall (\alpha_1 \rightarrow (\alpha_2 \rightarrow \alpha_1)) > \tau_7 \Rightarrow \tau_8 \rightarrow \tau_1 \]

So \[ \left\{ \begin{array}{l}
\tau_8 = \tau_9 \\
\tau_1 = (\tau_{10} \rightarrow \tau_9)
\end{array} \right. \text{ for some } \tau_9, \tau_{10}, \tau_{11}, \tau_{12} \]

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 } 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 } 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

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

give

\[\tau_7 \Rightarrow (C13) \ \tau_4 \Rightarrow (C12) \ \tau_6 \Rightarrow (C11) \ \tau_7 \Rightarrow \tau_5\]
The constraints generated from trying to type

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

give

\[\tau_7 \overset{(C13)}{=} \tau_4 \overset{(C12)}{=} \tau_6 \overset{(C11)}{=} \tau_7 \rightarrow \tau_5\]

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

\[(a) \vdash M : \forall A (\tau) \quad \text{(i.e. } \{\} \vdash M : \tau \text{ is provable)} \quad \& \quad A = \text{f\texttt{tv}} (\tau)\]
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 $\forall A \; (\tau) \prec \tau'$
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 $\forall A (\tau) \succ \tau'$

e.g. $\forall \alpha_1, \alpha_2, \alpha_3 (\alpha_1 \to (\alpha_2 \to (\alpha_3 \to \alpha_2)))$ 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 (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 $M$.

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

\[
\begin{align*}
\text{let } & \text{pair} = \lambda x \lambda y \lambda z (z \times y) ) \text{ in} \\
\text{let } & x_1 = \lambda y (\text{pair} y y) \text{ in} \\
\text{let } & x_2 = \lambda y (x_1 (x_1 y)) \text{ in} \\
\text{let } & x_3 = \lambda y (x_2 (x_2 y)) \text{ in} \\
\text{let } & x_4 = \lambda y (x_3 (x_3 y)) \text{ in} \\
\text{let } & x_5 = \lambda y (x_4 (x_4 y)) \text{ in} \\
& x_5 (\lambda y (y))
\end{align*}
\]
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

\[
S(\tau_1) = S(\tau_2).
\]
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) \( S(\tau_1) = S(\tau_2) \).
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 unifiable, i.e. whether there exists a type-substitution \(S \in \text{Sub}\) with

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

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

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

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

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

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

(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 $\Gamma \vdash M : \tau$ 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 \( \Gamma = \{ x_1 : \sigma_1, \ldots, x_n : \sigma_n \} \)).
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 \(T(\sigma) \succeq \sigma'\).
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 $T(\sigma) \succ \sigma'$.

(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 $\sigma \succ \tau'$.)
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 $T(\sigma) \succ \sigma'$.

(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 $\sigma \succ \tau'$.)

eg $\forall \alpha (\alpha \rightarrow \beta) \succ (\beta \rightarrow \beta) \rightarrow \beta$

but $\forall \alpha (\alpha \rightarrow \beta) \nleq \forall \beta ((\beta \rightarrow \beta) \rightarrow \beta)$
Example typing problem and solutions

Typing problem

```
x : ∀α (β → (γ → α)) ⊢ x true : ?
```

has solutions:

\[ I_S^1 = \{ b \}, s_1 = 8a(ba) \]
\[ I_S^2 = \{ b, g \}, s_2 = 8a0(aa0) \]
\[ I_S^3 = \{ b, g \}, s_3 = 8a0(0a0) \]
\[ I_S^4 = \{ b, g \}, s_4 = 8\{\} \]

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:

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

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

\[
\Delta \quad \sum_2 \left( \forall \alpha (\beta \rightarrow (\gamma \rightarrow \alpha)) = \forall \alpha' (\text{bool} \rightarrow (\alpha \rightarrow \alpha')) \right) \quad \text{(any } \alpha' \not= \alpha)\]
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

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

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 \((S_1, s_1)\) 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:

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

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 $\sigma \succ \sigma'$, 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 \( M \)).

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 \( A = (\text{ftv}(\tau) \ominus \text{ftv}(S \Gamma)) \), 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 applications: $pt(\Gamma \vdash M_1 M_2 : ?) \triangleq$

let $(S_1, \tau_1) = pt(\Gamma \vdash M_1 : ?)$ in
let $(S_2, \tau_2) = pt(S_1 \Gamma \vdash M_2 : ?)$ in
let $\alpha = \text{fresh}$ in
let $S_3 = mgu(S_2 \tau_1, \tau_2 \rightarrow \alpha)$ in $(S_3 S_2 S_1, S_3(\alpha))$
\( \text{pt}(\Gamma \vdash M_1 : ?) = (S_1, \tau_1) \)

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

\[ \downarrow \quad \text{pt}(\Gamma' \vdash M_1 M_2 : ?) = \]
\[ \mathsf{pt}(\Gamma \vdash M_1 : ?) = (S_1, \tau_1) \]

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

\[ S_2S_1 \Gamma \vdash M_1 : S_2 \tau_1 \]

\[ S_2S_1 \Gamma' \vdash M_2 : \tau_2 \]

\[ \downarrow \]

\[ \mathsf{pt}(\Gamma' \vdash M_1 M_2 : ?) = \]
\[ pt(\Gamma \vdash M_1 : ?) = (S_1, \tau_1) \]

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

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

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

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

\[ \Rightarrow pt(\Gamma' \vdash M_1 M_2 : ?) = (S_3 S_2 S_1, S_3 \alpha) \]
\[ p_t(\Gamma \vdash M_1 : ?) = (S_1, \tau_1) \]
\[ p_t(S_1 \Gamma \vdash M_2 : ?) = (S_2, \tau_2) \]
\[ \text{mgu}(S_2 \tau_1, \tau_2 \to \alpha) = S_3 \]

\[ S_3S_2S_1 \Gamma \vdash M_1 : S_3 \tau_2 \to S_3 \alpha \]
\[ S_3S_2S_1 \Gamma' \vdash M_2 : S_3 \tau_2 \]

\[ \frac{S_3S_2S_1 \Gamma \vdash M_1 M_2 : S_3 \alpha}{(\text{app})} \]

\[ p_t(\Gamma' \vdash M_1 M_2 : ?) = (S_3S_2S_1, S_3 \alpha) \]