PLC type system

(var) \[\Gamma \vdash x : \tau \text{ if } (x : \tau) \in \Gamma\]

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

(app) \[\Gamma \vdash M : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash M' : \tau_1 \Rightarrow \Gamma \vdash MM' : \tau_2\]

(gen) \[\Gamma \vdash M : \tau \Rightarrow \Gamma \vdash \Lambda \alpha (M) : \forall \alpha (\tau) \text{ if } \alpha \notin \text{ftv}(\Gamma)\]

(spec) \[\Gamma \vdash M : \forall \alpha (\tau_1) \Rightarrow \Gamma \vdash M \tau_2 : \tau_1[\tau_2/\alpha]\]
PLC operator association

\[ M_1 M_2 M_3 \text{ means } (M_1 M_2) M_3 \]

\[ M_1 M_2 \tau \text{ means } (M_1 M_2) \tau, \text{ etc.} \]

\[ \forall \alpha_1, \alpha_2 (\tau) \text{ means } \forall \alpha_1 (\forall \alpha_2 (\tau)) \]

\[ \lambda x_1: i_1, x_2: i_2 (M) \text{ means } \lambda x_1: i_1 (\lambda x_2: i_2 (M)) \]

\[ \land \alpha_1, \alpha_2 (M) \text{ means } \land \alpha_1 (\land \alpha_2 (M)) \]
Datatypes in PLC [Sect. 4.4]

- define a suitable PLC type for the data
- define suitable PLC expressions for values & operations on the data
- show PLC expressions have correct typings & computational behaviour

need to give PLC an operational semantics
Functions on types

In PLC, $\Lambda\alpha (M)$ is an anonymous notation for the function $F$ mapping each type $\tau$ to the value of $M[\tau/\alpha]$ (of some particular type).
Functions on types

In PLC, $\Lambda \alpha (M)$ is an anonymous notation for the function $F$ mapping each type $\tau$ to the value of $M[\tau/\alpha]$ (of some particular type).

$F \tau$ denotes the result of applying such a function to a type.
Functions on types

In PLC, $\Lambda \alpha (M)$ is an anonymous notation for the function $F$ mapping each type $\tau$ to the value of $M[\tau/\alpha]$ (of some particular type).

$F \tau$ denotes the result of applying such a function to a type.

Computation in PLC involves beta-reduction for such functions on types

$$(\Lambda \alpha (M)) \tau \rightarrow M[\tau/\alpha]$$
Functions on types

In PLC, $\Lambda \alpha (M)$ is an anonymous notation for the function $F$ mapping each type $\tau$ to the value of $M[\tau/\alpha]$ (of some particular type).

$F \tau$ denotes the result of applying such a function to a type.

Computation in PLC involves beta-reduction for such functions on types

$$(\Lambda \alpha (M)) \tau \to M[\tau/\alpha]$$

as well as the usual form of beta-reduction from $\lambda$-calculus

$$(\lambda x : \tau (M_1)) M_2 \to M_1[M_2/x]$$
Beta-reduction of PLC expressions

\( M \text{ beta-reduces to } M' \text{ in one step, } M \rightarrow M' \text{ means } M' \text{ can be obtained from } M \text{ (up to alpha-conversion, of course) by replacing a subexpression which is a redex by its corresponding reduct. } \)

The redex-reduct pairs are of two forms:

\[
(\lambda x : \tau (M_1)) M_2 \rightarrow M_1[M_2/x]
\]

\[
(\Lambda \alpha (M)) \tau \rightarrow M[\tau/\alpha]
\]

\( M_1[M_2/\alpha] = \text{result of substituting } M_2 \text{ for all free occurrences of } \alpha \text{ in } M_1 \) (avoiding capture of free vars & tvars in \( M_2 \) by binders in \( M_1 \))

\( M[\tau/\alpha] = \text{result of substituting } \tau \text{ for all free occurrences of } \alpha \text{ in } M \) (avoiding capture)
\[(\lambda x: \alpha_1 \to \alpha_1 (xy)) \ ( (\forall \alpha_2 (\lambda z: \alpha_2 (z)) (\alpha_1 \to \alpha_1 )) \)
(\lambda x : \alpha_1 \to \alpha_1 \ (xy)) \ (\lambda \alpha_2 (\lambda z : \alpha_2 (z)) \ (\alpha_1 \to \alpha_4))

(\lambda z : \alpha_1 \to \alpha_1 (z))
\[(\lambda x: \alpha, \neg \alpha_1 (xy)) \quad (\land \alpha_2 (\lambda z: \alpha_2 (z))) \quad (\alpha, \neg \alpha_1)\]

\[(\land \alpha_2 (\lambda z: \alpha_2 (z))) \quad (\lambda z: \alpha, \neg \alpha_1 (z))\]

\[(\lambda z: \alpha, \neg \alpha_1 (z)) \quad y\]
\[ (\lambda x : \alpha_1 \to \alpha_1 \, (xy)) \quad (\forall \alpha_2 (\lambda z : \alpha_2 (z))) (\alpha_1 \to \alpha_1) \]

\[ (\lambda x : \alpha_1 \to \alpha_1 (xy)) (\lambda z : \alpha_1 \to \alpha_1 (z)) y \]

\[ (\forall \alpha_2 (\lambda z : \alpha_2 (z))) (\alpha_1 \to \alpha_1) y \]

\[ (\lambda z : \alpha_1 \to \alpha_1 (z)) y \]
\[ p^{44} \]

\[
(\lambda x : \alpha_1 \to \alpha_1 (x y)) \quad (\forall \alpha_2 (\lambda z : \alpha_2 (z)) (\alpha_1 \to \alpha_1))
\]

\[
(\lambda x : \alpha_1 \to \alpha_1 (x y)) (\lambda z : \alpha_1 \to \alpha_1 (z))
\]

\[
(\forall \alpha_2 (\lambda z : \alpha_2 (z)) (\alpha_1 \to \alpha_1))
\]

\[
(\lambda z : \alpha_1 \to \alpha_1 (z)) y
\]
\((\lambda x: \alpha_1 \to \alpha_1 (xy)) \quad (\forall \alpha_2 (\lambda z: \alpha_2 (z)))(\alpha_1 \to \alpha_1)\)

\((\lambda x: \alpha_1 \to \alpha_1 (xy)) (\lambda z: \alpha_1 \to \alpha_1 (z))\)

\((\forall \alpha_2 (\lambda z: \alpha_2 (z)))(\alpha_1 \to \alpha_1)\) \quad \text{y}

\((\lambda z: \alpha_1 \to \alpha_1 (z)) \text{y}\)
Beta-reduction of PLC expressions

\( M \) beta-reduces to \( M' \) in one step, \( M \rightarrow M' \) means \( M' \) can be obtained from \( M \) (up to alpha-conversion, of course) by replacing a subexpression which is a redex by its corresponding reduct. The redex-reduct pairs are of two forms:

\[
(\lambda x : \tau (M_1)) M_2 \rightarrow M_1[M_2/x]
\]

\[
(\Lambda \alpha (M)) \tau \rightarrow M[\tau/\alpha]
\]

\( M \rightarrow^* M' \) indicates a chain of finitely\( ^\dagger \) many beta-reductions.
Beta-reduction of PLC expressions

\( M \) beta-reduces to \( M' \) in one step, \( M \rightarrow M' \) means \( M' \) can be obtained from \( M \) (up to alpha-conversion, of course) by replacing a subexpression which is a redex by its corresponding reduct. The redex-reduct pairs are of two forms:

\[
(\lambda x : \tau (M_1)) M_2 \rightarrow M_1[M_2/x]
\]

\[
(\Lambda \alpha (M)) \tau \rightarrow M[\tau/\alpha]
\]

\( M \rightarrow^* M' \) indicates a chain of finitely\(^\dagger\) many beta-reductions.

(\( ^\dagger \) possibly zero – which just means \( M \) and \( M' \) are alpha-convertible).
Beta-reduction of PLC expressions

\( M \) beta-reduces to \( M' \) in one step, \( M \rightarrow M' \) means \( M' \) can be obtained from \( M \) (up to alpha-conversion, of course) by replacing a subexpression which is a redex by its corresponding reduct. The redex-reduct pairs are of two forms:

\[
(\lambda x : \tau (M_1)) M_2 \rightarrow M_1[M_2/x]
\]

\[
(\Lambda \alpha (M)) \tau \rightarrow M[\tau/\alpha]
\]

\( M \rightarrow^* M' \) indicates a chain of finitely\(^*\) many beta-reductions.

\(^*\) possibly zero – which just means \( M \) and \( M' \) are alpha-convertible).

\( M \) is in beta-normal form if it contains no redexes.
Properties of PLC beta-reduction on typeable expressions

Suppose $\Gamma \vdash M : \tau$ is provable in the PLC type system. Then the following properties hold:

**Subject Reduction.** If $M \rightarrow M'$, then $\Gamma \vdash M' : \tau$ is also a provable typing.
Subject reduction: if \( \Gamma \vdash M : 2 \) & \( M \rightarrow M' \), then \( \Gamma \vdash M' : 2 \)

\[
\begin{align*}
\Gamma, x : 2 \vdash M : 2 \\
\Gamma, x : 2 \vdash (\lambda x : 2' (M) : 2 \rightarrow 2') \Gamma \vdash M' : 2' \\
\Gamma, (\lambda x : 2' (M)) : 2' \vdash M' : 2
\end{align*}
\]
\[
\frac{\Gamma, x : 2 \vdash M : 2}{\Gamma \vdash \lambda x : 2 \, (\text{\texttt{M}}) : 2 \\rightarrow^r 2} \quad \frac{\Gamma \vdash M' : 2'}{\Gamma \vdash (\lambda x : 2 \, (\text{\texttt{M}})) \, M' : 2}
\]
\[\downarrow_\beta\]
\[M[M'/x]\]
\[
\Gamma, x : 2 \vdash M : 2 \\
\Gamma \vdash \lambda x : 2' (\lambda x : 2' (M) : 2 \to 2) : 2' \\
\Gamma \vdash \lambda x : 2' (\lambda x : 2' (M) : 2 \to 2) M' : 2
\]

\[\downarrow_{\beta} M[M'/x] \leftarrow \text{to see that this has type } 2, \text{ need to prove a Substitution Lemma} \]
If \( \Gamma \vdash M : 2 \) and \( \Gamma \vdash M' : 2' \)

then

\( \Gamma \vdash M[M'/x] : 2 \)

**Substitution Lemma**

(proved by induction on structure of \( M \))
Subject reduction: if $\Gamma \vdash M : \tau$ & $M \rightarrow M'$, then $\Gamma \vdash M' : \tau$.
\[
\frac{
    \Gamma \vdash \text{w} : 2 \\
    \alpha \notin \text{fv}(\Gamma)
}{
\Gamma \vdash \text{v} : 2 (\forall \alpha (\text{w}) \exists' z \; z' \rightarrow z)
}
\]
\[
\downarrow_{\beta}
\]
\[
M[z'/\alpha]
\]

To see that this has type \( 2 \rightarrow 2 \rightarrow 2 \), need to prove a Substitution Lemma.
If \( \Gamma \vdash w : 2 \) \& \( \alpha \in \text{fv}(\Gamma) \)

then

\[ \Gamma, \alpha \vdash [\alpha \mapsto w] \]

(proved by induction of structure of \( \Gamma \))

Substitution Lemma
Properties of PLC beta-reduction on typeable expressions

Suppose $\Gamma \vdash M : \tau$ is provable in the PLC type system. Then the following properties hold:

**Subject Reduction.** If $M \rightarrow M'$, then $\Gamma \vdash M' : \tau$ is also a provable typing.

**Church Rosser Property.** If $M \rightarrow^* M_1$ and $M \rightarrow^* M_2$, then there is $M'$ with $M_1 \rightarrow^* M'$ and $M_2 \rightarrow^* M'$.
Properties of PLC beta-reduction on typeable expressions

Suppose $\Gamma \vdash M : \tau$ is provable in the PLC type system. Then the following properties hold:

**Subject Reduction.** If $M \rightarrow M'$, then $\Gamma \vdash M' : \tau$ is also a provable typing.

**Church Rosser Property.** If $M \rightarrow^* M_1$ and $M \rightarrow^* M_2$, then there is $M'$ with $M_1 \rightarrow^* M'$ and $M_2 \rightarrow^* M'$.

**Strong Normalisation Property.** There is no infinite chain $M \rightarrow M_1 \rightarrow M_2 \rightarrow \ldots$ of beta-reductions starting from $M$. 
Properties of PLC beta-reduction on typeable expressions

Suppose $\Gamma \vdash M : \tau$ is provable in the PLC type system. Then the following properties hold:

**Subject Reduction.** If $M \rightarrow M'$, then $\Gamma \vdash M' : \tau$ is also a provable typing.

**Church Rosser Property.** If $M \rightarrow^* M_1$ and $M \rightarrow^* M_2$, then there is $M'$ with $M_1 \rightarrow^* M'$ and $M_2 \rightarrow^* M'$.

**Strong Normalisation Property.** There is no infinite chain $M \rightarrow M_1 \rightarrow M_2 \rightarrow \ldots$ of beta-reductions starting from $M$.

$\Omega \triangleq (\lambda x: \alpha (xx))(\lambda x: \alpha (xx))$ satisfies $\Omega \rightarrow \Omega \rightarrow \Omega \rightarrow \ldots$

but it's not typeable (nor is the fixpoint combinator, $Y$)
Theorem 15: (p. 46)

Church Rosser (CR) + Strong Normalization (SN) 

⇒ Exist unique beta-normal forms for typeable PLC expressions

Existence: start from M & reduce any old way ... must eventually stop by SN

Uniqueness: if \( M \) 

\[ \star N_1 \rightarrow \star N_2 \rightarrow \]
Theorem 15. (p. 46)

Church Rosser (CR) + Strong Normalization (SN)

⇒ Exist unique beta-normal forms for typeable PLC expressions

Existence: start from M & reduce any old way... must eventually stop by SN

Uniqueness: if M \* N₁ \* M' by CR

N₁ \* N₂
Theorem 15 (p46)

Church Rosser (CR) + Strong Normalization (SN)

⇒ \textbf{Exist unique beta-normal forms for typeable PLC expressions}

Existence: start from M & reduce any old way... must eventually stop by SN

Uniqueness: if M \rightarrow^* N_1 \rightarrow^* M' so N_1 \rightarrow^* M \rightarrow^* M' \alpha\text{-equiv} N_2 \rightarrow^* N_1 \rightarrow^* M' \alpha\text{-equiv} N_2
PLC beta-conversion, $=\beta$

By definition, $M =\beta M'$ holds if there is a finite chain
PLC beta-conversion, $=_{\beta}$

By definition, $M =_{\beta} M'$ holds if there is a finite chain

$$M \rightarrow \cdots \rightarrow M'$$

where each $\rightarrow$ is either $\rightarrow$ or $\leftarrow$, i.e. a beta-reduction in one direction or the other.
PLC beta-conversion, $=_{\beta}$

By definition, $M =_{\beta} M'$ holds if there is a finite chain

$$M \rightarrow \cdots \rightarrow M'$$

where each $\rightarrow$ is either $\rightarrow$ or $\leftarrow$, i.e. a beta-reduction in one direction or the other. (A chain of length zero is allowed—in which case $M$ and $M'$ are equal, up to alpha-conversion, of course.)

So $=_{\beta}$ is the smallest equivalence relation containing $\rightarrow$.
PLC beta-conversion, $=\beta$

By definition, $M =_\beta M'$ holds if there is a finite chain

$$M \rightarrow \cdots \rightarrow \cdots \rightarrow \cdots \rightarrow M'$$

where each $\rightarrow$ is either $\rightarrow$ or $\leftarrow$, i.e. a beta-reduction in one direction or the other. (A chain of length zero is allowed—in which case $M$ and $M'$ are equal, up to alpha-conversion, of course.)

Church Rosser + Strong Normalisation properties imply that, for typeable PLC expressions, $M =_\beta M'$ holds if and only if there is some beta-normal form $N$ with

$$M \rightarrow^* N \leftarrow^* M'$$
Datatypes in PLC  [Sect. 4.4]

- define a suitable PLC type for the data
- define suitable PLC expressions for values & operations on the data
- show PLC expressions have correct typings & computational behaviour
Polymorphic booleans

\[ bool \triangleq \forall \alpha (\alpha \to (\alpha \to \alpha)) \]
Polymorphic booleans

\[ bool \triangleq \forall \alpha (\alpha \to (\alpha \to \alpha)) \]

In ML/Haskell/Scala/..., have
datatype bool = True | False
and each \( B : \text{bool} \) gives us a polymorphic function
\[ \lambda x,y. \text{if } B \text{ then } x \text{ else } y : \forall \alpha (\alpha \to \alpha \to \alpha) \]
Polymorphic booleans

\[\text{bool} \triangleq \forall \alpha (\alpha \rightarrow (\alpha \rightarrow \alpha))\]

In ML/Haskell/Scala/... have datatype `bool = True | False` and each \(B : \text{bool}\) gives us a polymorphic function \(\lambda x,y. \text{if } B \text{ then } x \text{ else } y : \forall \alpha (\alpha \rightarrow \alpha \rightarrow \alpha)\)

**IDEA**: identify Booleans with expressions of this type.
Polymorphic booleans

\[\text{bool} \triangleq \forall \alpha (\alpha \to (\alpha \to \alpha))\]

\[\text{True} \triangleq \Lambda \alpha (\lambda x_1 : \alpha, x_2 : \alpha (x_1))\]

\[\text{False} \triangleq \Lambda \alpha (\lambda x_1 : \alpha, x_2 : \alpha (x_2))\]

\{ \} \vdash \text{True : bool} \\
\{ \} \vdash \text{False : bool}
Polymorphic booleans

\[ bool \triangleq \forall \alpha (\alpha \to (\alpha \to \alpha)) \]

\[ True \triangleq \Lambda \alpha (\lambda x_1 : \alpha, x_2 : \alpha (x_1)) \]

\[ False \triangleq \Lambda \alpha (\lambda x_1 : \alpha, x_2 : \alpha (x_2)) \]

\[ if \triangleq \Lambda \alpha (\lambda b : bool, x_1 : \alpha, x_2 : \alpha (b \alpha x_1 x_2)) \]

\[ \{ \} \vdash if : \forall \alpha (bool \to (\alpha \to (\alpha \to \alpha))) \]
If \( \{ M_1 \rightarrow^* \text{true} \} \), then

\[
\text{if } \exists M_1 \, M_2 \, M_3 \rightarrow^* \text{ if } \tau \text{ true } M_2 \, M_3
\]
If \( \{ M_1 \rightarrow^{*} \text{Tme} \), then

\[
\text{if } \tau \vdash M_1 M_2 M_3 \rightarrow^{*} \text{if } \tau \vdash \text{Tme} M_2 M_3 \parallel \Lambda\alpha(\ldots) \vdash \text{Tme} M_2 M_3
\]
If \( \{ M_1 \rightarrow* \text{Tme} \}
\begin{cases} \quad \text{and} \\ M_2 \rightarrow* N \end{cases} \), then

\[
\text{if } \tau \rightarrow M_1 M_2 M_3 \rightarrow* \text{ if } \tau \rightarrow \text{Tme} M_2 M_3 \\
\downarrow \\
\forall \alpha(...) \rightarrow \text{Tme} M_2 M_3 \\
\downarrow \\
(\lambda a : \text{bool}, x_1 : \text{int}, x_2 : \text{int} \rightarrow (b \rightarrow x_1 x_2)) \rightarrow \text{Tme} M_2 M_3
\]
If \( \{ M_1 \rightarrow^* \mathit{True} \), then

\[
\text{if } \tau \triangleright M_1 M_2 M_3 \rightarrow^* \text{ if } \tau \mathit{True} \ {M_2} \ {M_3} \\
\downarrow \downarrow \downarrow \\
\mathit{True}(\ldots) \triangleright \mathit{True} \ {M_2} \ {M_3} \\
\downarrow \\
\mathit{True} \ {M_2} \ {M_3} \\
\downarrow \\
\mathit{True} \ {\tau} \ {M_2} \ {M_3}
\]
If \( \{ M_1 \rightarrow^* \text{Tme}, \text{then} \}
\]

\[
\text{if } \tau \vdash M_1 M_2 M_3 \rightarrow^* \text{ if } \tau \vdash \text{Tme } M_2 M_3
\]

\[
\vdash \Lambda \alpha(\ldots) \vdash \text{Tme } M_2 M_3
\]

\[
(\lambda b : \text{body}, x_1 : \tau, x_2 : \tau (b \tau x_1 x_2)) \vdash \text{Tme } M_2 M_3
\]

\[
\downarrow
\]

\[
\Lambda \alpha (\lambda x_1 : \alpha, x_2 : \alpha(x_1)) \vdash M_2 M_3
\]

\[
\downarrow
\]

\[
\text{Tme } \tau M_2 M_3
\]

\[
\downarrow
\]

\[
\Lambda \alpha (\lambda x_1 : \alpha, x_2 : \alpha(x_1)) \vdash M_2 M_3
\]
If \( \{ M_1 \rightarrow^* \text{Tme} \), then

\[
\begin{align*}
\text{if } & \exists M_1, M_2, M_3 \rightarrow^* \text{ if } & \tau \text{Tme } M_2 \text{ M}_3 \\
& & \Rightarrow \\
& \Lambda \alpha(\ldots) \tau \text{Tme } M_2 \text{ M}_3 \\
& \downarrow \\
& (\lambda b : \text{bool}, x_1 : \tau, x_2 : \tau (b \tau x_1 x_2)) \text{Tme } M_2 \text{ M}_3 \\
& \downarrow \\
& \lambda x \text{Tme } \tau M_2 \text{ M}_3 \\
& \downarrow \\
& \Lambda \alpha(\lambda x_1 : \alpha, x_2 : \alpha(x_1)) \tau M_2 \text{ M}_3
\end{align*}
\]
**FACT:** \[ \text{True} \triangleq \Lambda \alpha (\lambda x_1, x_2 : \alpha (x_1)) \]

\[ \text{False} \triangleq \Lambda \alpha (\lambda x_1, x_2 : \alpha (x_2)) \]

are the **only** closed expressions in \( \beta \)-normal form of type \( \text{bool} \triangleq \forall \alpha (\alpha \rightarrow (\alpha \rightarrow \alpha)) \).