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

Example: finite lists [p48 →]

Iteratively defined functions on finite lists

$A^* \triangleq$ finite lists of elements of the set A

Notation :

empty list : $\mathrm{Nil} \in A^*$

cons : $\dfrac{x \in A \qquad \ell \in A^*}{x :: \ell \in A^*}$

Iteratively defined functions on finite lists

$A^* \triangleq$ finite lists of elements of the set A

Given a set B, an element $x' \in B$, and a function $f : A \to B \to B$, the *iteratively defined function* $\mathit{listIter}\, x'\, f$ is the unique function $g : A^* \to B$ satisfying:

$$g\, \mathit{Nil} = x'$$
$$g\, (x :: \ell) = f\, x\, (g\, \ell)$$

for all $x \in A$ and $\ell \in A^*$.

Iteratively defined functions on finite lists

$A^* \triangleq$ finite lists of elements of the set A

Given a set B, an element $x' \in B$, and a function $f : A \to B \to B$, the *iteratively defined function* $\mathit{listIter}\, x'\, f$ is the unique function $g : A^* \to B$ satisfying:

$$g\,\mathit{Nil} = x'$$
$$g\,(x :: \ell) = f\,x\,(g\,\ell)$$

for all $x \in A$ and $\ell \in A^*$.

$$g\,\mathrm{Nil} = x'$$
$$g\,(x_1 :: \mathrm{Nil}) = f\,x_1\,x'$$
$$g\,(x_2 :: x_1 :: \mathrm{Nil}) = f\,x_2\,(f\,x_1\,x')$$
$$g\,(x_n :: \cdots :: x_1 :: \mathrm{Nil}) = f\,x_n\,(\cdots(f\,x_1\,x')\cdots)$$

Iteratively defined functions on finite lists

$A^* \triangleq$ finite lists of elements of the set A

Given a set B, an element $x' \in B$, and a function $f : A \to B \to B$, the *iteratively defined function* $\mathit{listIter}\, x'\, f$ is the unique function $g : A^* \to B$ satisfying:

$$g\, \mathit{Nil} = x'$$
$$g\, (x :: \ell) = f\, x\, (g\, \ell)$$

for all $x \in A$ and $\ell \in A^*$.

For each $\ell \in A^*$ is a function which is "polymorphic" in B (& A)

$$\boxed{x', f \longmapsto \mathit{listIter}\, x'\, f}$$

$$B \to (A \to B \to B) \to B$$

Polymorphic lists

$$\alpha\ list \triangleq \forall \alpha' \, (\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha')$$

Polymorphic lists

$$\alpha\ list \triangleq \forall \alpha' \,(\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha')$$

$$Nil \triangleq \Lambda \alpha, \alpha' \,(\lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha' \,(x'))$$

Polymorphic lists

$$\alpha\ list \triangleq \forall\alpha'\ (\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha')$$

$$Nil \triangleq \Lambda\alpha, \alpha'\ (\lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha'\ (x'))$$

$$Cons \triangleq \Lambda\alpha(\lambda x : \alpha, \ell : \alpha\ list(\Lambda\alpha'($$

Polymorphic lists

$$\alpha\; list \triangleq \forall \alpha' \,(\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha')$$

$$Nil \triangleq \Lambda \alpha, \alpha' \,(\lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha' \,(x'))$$

$$\begin{aligned} Cons \triangleq \Lambda \alpha (\lambda x : \alpha, \ell : \alpha\; list (\Lambda \alpha' (\\ \lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha' (\\ f\, x\, (\ell\, \alpha'\, x'\, f))))) \end{aligned}$$

Polymorphic lists

$: \forall \alpha\, (\alpha\, list)$

$$\alpha\; list \triangleq \forall \alpha'\, (\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha')$$

$$Nil \triangleq \Lambda \alpha, \alpha'\, (\lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha'\, (x'))$$

$$\begin{aligned} Cons \triangleq \Lambda \alpha (\lambda x : \alpha, \ell : \alpha\; list (\Lambda \alpha' (\\ \lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha' (\\ f\, x\, (\ell\, \alpha'\, x'\, f))))) \end{aligned}$$

$: \forall \alpha\, (\alpha \to \alpha\, list \to \alpha\, list)$

List iteration in PLC

$$
\begin{gathered}
iter \triangleq \Lambda \alpha, \alpha' (\lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha' (\\
\lambda \ell : \alpha\, list\, (\ell\, \alpha'\, x'\, f)))
\end{gathered}
$$

satisfies:

- $\vdash iter : \forall \alpha, \alpha'\, (\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha\, list \to \alpha')$

List iteration in PLC

$$iter \triangleq \Lambda\alpha, \alpha' (\lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha' (\\ \lambda \ell : \alpha\, list\, (\ell\, \alpha'\, x'\, f)))$$

satisfies:

- $\vdash iter : \forall \alpha, \alpha' \, (\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha\, list \to \alpha')$
- $iter\, \alpha\, \alpha'\, x'\, f\, (Nil\, \alpha) =_\beta x'$

List iteration in PLC

$$iter \triangleq \Lambda\alpha, \alpha' (\lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha' ($$
$$\lambda \ell : \alpha\, list\, (\ell\, \alpha'\, x'\, f)))$$

satisfies:

- $\vdash iter : \forall \alpha, \alpha' \, (\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha\, list \to \alpha')$
- $\boxed{iter\, \alpha\, \alpha'\, x'\, f\, (Nil\, \alpha)} =_\beta x'$

$\downarrow *$

$\boxed{Nil\, \alpha\, \alpha'\, x'\, f}$

$*$

List iteration in PLC

$$iter \triangleq \Lambda \alpha, \alpha' \,(\lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha' \,(\\ \lambda \ell : \alpha\, list\, (\ell\, \alpha'\, x'\, f)))$$

satisfies:

- $\vdash iter : \forall \alpha, \alpha' \,(\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha\, list \to \alpha')$
- $iter\, \alpha\, \alpha'\, x'\, f\, (Nil\, \alpha) =_\beta x'$
- $iter\, \alpha\, \alpha'\, x'\, f\, (Cons\, \alpha\, x\, \ell) =_\beta f\, x\, (iter\, \alpha\, \alpha'\, x'\, f\, \ell)$

List iteration in PLC

$$iter \triangleq \Lambda\alpha, \alpha' (\lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha' (\\ \lambda \ell : \alpha\, list\, (\ell\, \alpha'\, x'\, f)))$$

satisfies:

- $\vdash iter : \forall \alpha, \alpha' \, (\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha\, list \to \alpha')$
- $iter\, \alpha\, \alpha'\, x'\, f\, (Nil\, \alpha) =_\beta x'$
- $\boxed{iter\, \alpha\, \alpha'\, x'\, f\, (Cons\, \alpha\, x\, \ell)} =_\beta f\, x\, (iter\, \alpha\, \alpha'\, x'\, f\, \ell)$

$\xrightarrow{*} (Cons\, \alpha\, x\, \ell)\, \alpha'\, x'\, f \to^{*} f\, x\, (\ell\, \alpha'\, x'\, f)$

List iteration in PLC

$$iter \triangleq \Lambda \alpha, \alpha' (\lambda x' : \alpha', f : \alpha \to \alpha' \to \alpha' (\\ \lambda \ell : \alpha\ list\ (\ell\ \alpha'\ x'\ f)))$$

satisfies:

- $\vdash iter : \forall \alpha, \alpha'\ (\alpha' \to (\alpha \to \alpha' \to \alpha') \to \alpha\ list \to \alpha')$
- $iter\ \alpha\ \alpha'\ x'\ f\ (Nil\ \alpha) =_\beta x'$
- $\boxed{iter\ \alpha\ \alpha'\ x'\ f\ (Cons\ \alpha\ x\ \ell)} =_\beta \boxed{f\ x\ (iter\ \alpha\ \alpha'\ x'\ f\ \ell)}$

$(Cons\ \alpha\ x\ \ell)\ \alpha'\ x'\ f \to^* f\ x\ (\ell\ \alpha'\ x'\ f)$

FACT Given a closed PLC type τ

$$\boxed{\begin{array}{l}\{\text{closed } \beta\text{-normal forms of type } \tau\,\text{list}\} \\ \cong \\ \{\text{closed } \beta\text{-normal forms of type } \tau\}^*\end{array}}$$

$$\text{nil} \longleftrightarrow \beta NF(Nil\,\tau)$$

$$N_1 :: \text{nil} \longleftrightarrow \beta NF(Cons\,\tau\,(N_1\,(Nil\,\tau)))$$

$$N_2 :: N_1 :: \text{nil} \longleftrightarrow \beta NF(Cons\,\tau\,(N_2\,(Cons\,\tau\,(N_1\,(Nil\,\tau)))))$$

etc

"Algebraic" data types in ML

$$\boxed{\texttt{datatype } (\alpha_1, \ldots, \alpha_n)\,\texttt{alg} = C_1 \texttt{ of } \tau_1 \mid \cdots \mid C_m \texttt{ of } \tau_m}$$

types $\tau_1 \cdots \tau_m$ built up from $\alpha_1 \ldots \alpha_n$ and the type $(\alpha_1 \ldots \alpha_n)\,\texttt{alg}$ using $\texttt{unit}$, $\_ * \_$ & previously declared alg. datatypes

"Algebraic" data types in ML

$$\boxed{\text{datatype } (\alpha_1, \ldots, \alpha_n)\,\text{alg} = C_1 \text{ of } \tau_1 \mid \ldots \mid C_m \text{ of } \tau_m}$$

Eg.

datatype bool = T of unit | F of unit

datatype α list = Nil of unit |
Cons of $\alpha * \alpha$ list

E.g. of a <u>non</u>-algebraic ML datatype

```
datatype nTree = Leaf
               | Node of (nat -> nTree)
```

[Fig.5, p50]

PLC encodings of ML algebraic datatypes

ML	PLC
$\alpha_1 * \alpha_2$	$\forall\alpha((\alpha_1 \to \alpha_2 \to \alpha) \to \alpha)$
datatype (α_1, α_2)sum = Inl of α_1 | Inr of α_2	$\forall\alpha((\alpha_1 \to \alpha) \to (\alpha_2 \to \alpha) \to \alpha)$
datatype nat = Zero | Succ of nat	$\forall\alpha(\alpha \to (\alpha \to \alpha) \to \alpha)$
datatype binTree = Leaf | Node of binTree * binTree	$\forall\alpha(\alpha \to (\alpha \to \alpha \to \alpha) \to \alpha)$

[Fig.5, p50]

PLC encodings of ML algebraic datatypes

ML	PLC
$\alpha_1 * \alpha_2$	$\forall\alpha((\alpha_1 \to \alpha_2 \to \alpha) \to \alpha)$
datatype (α_1, α_2)sum = Inl of α_1 | Inr of α_2	$\forall\alpha((\alpha_1 \to \alpha) \to (\alpha_2 \to \alpha) \to \alpha)$
datatype nat = Zero | Succ of nat	$\forall\alpha(\alpha \to (\alpha \to \alpha) \to \alpha)$
datatype binTree = Leaf | Node of binTree * binTree	$\forall\alpha(\alpha \to (\alpha \to \alpha \to \alpha) \to \alpha)$

Standard ML signatures and structures

```
signature QUEUE =
  sig
    type 'a queue
    exception Empty
    val empty : 'a queue
    val insert : 'a * 'a queue -> 'a queue
    val remove : 'a queue -> 'a * 'a queue
end

structure Queue =
  struct
    type 'a queue = 'a list * 'a list
    exception Empty
    val empty = (nil, nil)
    fun insert (f, (front,back)) = (f::front, back)
    fun remove (nil, nil) = raise Empty
      | remove (front, nil) = remove (nil, rev front)
      | remove (front, b::back) = (b, (front, back))
  end
```

PLC + existential types

Types

$$t ::= \cdots \mid \exists\, \alpha\, (\tau)$$

Expressions

$$\begin{aligned} M ::= \cdots \mid\ & \mathtt{pack}\,(\tau, M) : \exists\, \alpha\, (\tau) \mid \\ & \mathtt{unpack}\, M : \exists\, \alpha\, (\tau)\ \mathtt{as}\ (\alpha, x)\ \mathtt{in}\ M : \tau \end{aligned}$$

PLC + existential types

Types

$t ::= \cdots \mid \exists\,\alpha\,(\tau)$

Expressions

$M ::= \cdots \mid \mathtt{pack}\,(\tau, M) : \exists\,\alpha\,(\tau) \mid$
$\qquad \mathtt{unpack}\,M : \exists\,\alpha\,(\tau)\ \mathtt{as}\ (\alpha, x)\ \mathtt{in}\ M : \tau$

Typing rules

$$(\exists\textbf{intro})\ \frac{\Gamma \vdash M : \tau[\tau'/\alpha]}{\Gamma \vdash (\mathtt{pack}\,(\tau', M) : \exists\,\alpha\,(\tau)) : \exists\,\alpha\,(\tau)}$$

$$(\exists\textbf{elim})\ \frac{\Gamma \vdash E : \exists\,\alpha\,(\tau) \qquad \Gamma, x : \tau \vdash M' : \tau'}{\Gamma \vdash (\mathtt{unpack}\,E : \exists\,\alpha\,(\tau)\ \mathtt{as}\ (\alpha, x)\ \mathtt{in}\ M' : \tau') : \tau'}$$

if $\alpha \notin \mathit{ftv}(\Gamma, \tau')$

PLC + existential types

Types

$t ::= \cdots \mid \exists\, \alpha\, (\tau)$

Expressions

$M ::= \cdots \mid \mathtt{pack}\, (\tau, M) : \exists\, \alpha\, (\tau) \mid$
$\quad\quad \mathtt{unpack}\, M : \exists\, \alpha\, (\tau)\ \mathtt{as}\ (\alpha, x)\ \mathtt{in}\, M : \tau$

Typing rules

$$(\exists\mathbf{intro})\ \frac{\Gamma \vdash M : \tau[\tau'/\alpha]}{\Gamma \vdash (\mathtt{pack}\, (\tau', M) : \exists\, \alpha\, (\tau)) : \exists\, \alpha\, (\tau)}$$

$$(\exists\mathbf{elim})\ \frac{\Gamma \vdash E : \exists\, \alpha\, (\tau) \qquad \Gamma, x : \tau \vdash M' : \tau'}{\Gamma \vdash (\mathtt{unpack}\, E : \exists\, \alpha\, (\tau)\ \mathtt{as}\ (\alpha, x)\ \mathtt{in}\, M' : \tau') : \tau'}$$

if $\alpha \notin ftv(\Gamma, \tau')$

Reduction

$$\mathtt{unpack}\, (\mathtt{pack}\, (\tau', M) : \exists\, \alpha\, (\tau)) : \exists\, \alpha\, (\tau)\ \mathtt{as}\ (\alpha, x)\ \mathtt{in}\, M' : \tau' \to M'[\tau'/\alpha, M/x]$$

Existential types in PLC

$$\exists\,\alpha\,(\tau) \triangleq \forall\beta\,((\forall\alpha\,(\tau\to\beta))\to\beta)$$

$$\mathtt{pack}\,(\tau', M) : \exists\,\alpha\,(\tau) \triangleq \Lambda\beta\,(\lambda y : \forall\alpha\,(\tau\to\beta)\,(y\,\tau' M))$$

$$\mathtt{unpack}\,E : \exists\,\alpha\,(\tau)\;\mathtt{as}\;(\alpha, x)\;\mathtt{in}\,M' : \tau' \triangleq E\,\tau'(\Lambda\alpha\,(\lambda x : \tau\,(M')))$$

(where $\beta \notin ftv(\alpha\,\tau\,\tau'\,M\,M')$)

Existential types in PLC

$$\exists\,\alpha\,(\tau) \triangleq \forall\beta\,((\forall\alpha\,(\tau \to \beta)) \to \beta)$$

$$\texttt{pack}\,(\tau', M) : \exists\,\alpha\,(\tau) \triangleq \Lambda\beta\,(\lambda y : \forall\alpha\,(\tau \to \beta)\,(y\,\tau' M))$$

$$\texttt{unpack}\,E : \exists\,\alpha\,(\tau)\;\texttt{as}\;(\alpha, x)\;\texttt{in}\,M' : \tau' \triangleq E\,\tau'(\Lambda\alpha\,(\lambda x : \tau\,(M')))$$

(where $\beta \notin ftv(\alpha\,\tau\,\tau'\,M\,M')$)

These definitions satisfy the typing and reduction rules on the previous slide (exercise).