Deferring the Details and Deriving Programs

Liam O'Connor

UNSW Sydney, Australia

TyDe, August 2019

Dependently-typed Data Structures

Dependent types are great. They let us bake our data invariants into our data structures:

$$
\begin{array}{l}
\textsf{data}\ \textsf{OList}\ (m\ n : \mathbb{N}) : \textsf{Set}\ \textsf{where} \\
\quad \textsf{Nil} : (m \leq n) \to \textsf{OList}\ m\ n \\
\quad \textsf{Cons} : (x : \mathbb{N}) \to (m \leq x) \to \textsf{OList}\ x\ n \to \textsf{OList}\ m\ n
\end{array}
$$

Dependently-typed Data Structures

Dependent types are great. They let us bake our data invariants into our data structures:

$$\begin{array}{l}\mathsf{data}\ \mathsf{OList}\ (m\ n : \mathbb{N}) : \mathsf{Set}\ \mathsf{where}\\ \quad \mathsf{Nil} : (m \leq n) \to \mathsf{OList}\ m\ n\\ \quad \mathsf{Cons} : (x : \mathbb{N}) \to (m \leq x) \to \mathsf{OList}\ x\ n \to \mathsf{OList}\ m\ n\end{array}$$

$$\begin{array}{l}\mathsf{data}\ \mathsf{BST}\ (m\ n : \mathbb{N}) : \mathsf{Set}\ \mathsf{where}\\ \quad \mathsf{Leaf} : (m \leq n) \to \mathsf{BST}\ m\ n\\ \quad \mathsf{Branch} : (x : \mathbb{N}) \to \mathsf{BST}\ m\ x \to \mathsf{BST}\ x\ n \to \mathsf{BST}\ m\ n\end{array}$$

Hoare Logic

$$\{\varphi\}\ P\ \{\psi\} \equiv P : [\ \varphi\ ,\ \psi\ ]$$

```
data [_,_] : Assertion → Assertion → Set₁ where
```

Hoare Logic

$$\{\varphi\}\ P\ \{\psi\} \equiv P : [\ \varphi\ ,\ \psi\ ]$$

$$\textsf{data}\ [\_,\_] : \mathsf{Assertion} \rightarrow \mathsf{Assertion} \rightarrow \mathsf{Set}_1\ \textsf{where}$$

$$\mathsf{SEQ} : [\ \varphi\ ,\ \alpha\ ] \rightarrow [\ \alpha\ ,\ \psi\ ] \rightarrow [\ \varphi\ ,\ \psi\ ]$$

Hoare Logic

$$\{\varphi\}\ P\ \{\psi\} \equiv P : [\ \varphi\ ,\ \psi\ ]$$

data [_,_] : Assertion → Assertion → Set_1 where

SEQ : $[\ \varphi\ ,\ \alpha\ ] \to [\ \alpha\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ]$

CHO : $[\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ]$

Hoare Logic

$$\{\varphi\}\ P\ \{\psi\} \equiv P : [\ \varphi\ ,\ \psi\ ]$$

$$\mathsf{data}\ [\_,\_] : \mathsf{Assertion} \to \mathsf{Assertion} \to \mathsf{Set}_1\ \mathsf{where}$$

$$\mathsf{SEQ} : [\ \varphi\ ,\ \alpha\ ] \to [\ \alpha\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ]$$

$$\mathsf{CHO} : [\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ]$$

$$\mathsf{STAR} : [\ \varphi\ ,\ \varphi\ ] \to [\ \varphi\ ,\ \varphi\ ]$$

Hoare Logic

$$\{\varphi\}\ P\ \{\psi\} \equiv P : [\ \varphi\ ,\ \psi\ ]$$

$$\mathsf{data}\ [\_,\_] : \mathsf{Assertion} \to \mathsf{Assertion} \to \mathsf{Set}_1\ \mathsf{where}$$

$$\mathsf{SEQ} : [\ \varphi\ ,\ \alpha\ ] \to [\ \alpha\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ]$$

$$\mathsf{CHO} : [\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ]$$

$$\mathsf{STAR} : [\ \varphi\ ,\ \varphi\ ] \to [\ \varphi\ ,\ \varphi\ ]$$

$$\mathsf{GUARD} : (g : \mathsf{Assertion}) \to [\ (g \to \varphi)\ ,\ \varphi\ ]$$

Hoare Logic

$$\{\varphi\}\ P\ \{\psi\} \equiv P : [\ \varphi\ ,\ \psi\ ]$$

```
data [_,_] : Assertion → Assertion → Set₁ where
  SEQ   : [ φ , α ] → [ α , ψ ] → [ φ , ψ ]
  CHO   : [ φ , ψ ] → [ φ , ψ ] → [ φ , ψ ]
  STAR  : [ φ , φ ] → [ φ , φ ]
  GUARD : (g : Assertion) → [ (g → φ) , φ ]
  UPD   : (Σ: φ → Σ: ψ) → [ φ , ψ ]
```

Hoare Logic

$$\{\varphi\}\ P\ \{\psi\} \equiv P : [\ \varphi\ ,\ \psi\ ]$$

$$
\begin{array}{l}
\textsf{data}\ [\_,\_] : \textsf{Assertion} \to \textsf{Assertion} \to \textsf{Set}_1\ \textsf{where} \\
\quad \textsf{SEQ} : [\ \varphi\ ,\ \alpha\ ] \to [\ \alpha\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ] \\
\quad \textsf{CHO} : [\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ] \\
\quad \textsf{STAR} : [\ \varphi\ ,\ \varphi\ ] \to [\ \varphi\ ,\ \varphi\ ] \\
\quad \textsf{GUARD} : (g : \textsf{Assertion}) \to [\ (g \to \varphi)\ ,\ \varphi\ ] \\
\quad \textsf{UPD} : (\Sigma\!:\ \varphi \to \Sigma\!:\ \psi) \to [\ \varphi\ ,\ \psi\ ] \\
\textsf{CONS} : \Pi\!:\ (\varphi \to \psi) \to [\ \varphi\ ,\ \psi\ ]
\end{array}
$$

Hoare Logic

$$\{\varphi\}\ P\ \{\psi\} \equiv P : [\ \varphi\ ,\ \psi\ ]$$

$$
\begin{array}{l}
\textsf{data}\ [\_,\_] : \textsf{Assertion} \to \textsf{Assertion} \to \textsf{Set}_1\ \textsf{where} \\
\quad \textsf{SEQ} : [\ \varphi\ ,\ \alpha\ ] \to [\ \alpha\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ] \\
\quad \textsf{CHO} : [\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ] \\
\quad \textsf{STAR} : [\ \varphi\ ,\ \varphi\ ] \to [\ \varphi\ ,\ \varphi\ ] \\
\quad \textsf{GUARD} : (g : \textsf{Assertion}) \to [\ (g \to \varphi)\ ,\ \varphi\ ] \\
\quad \textsf{UPD} : (\Sigma\!:\ \varphi \to \Sigma\!:\ \psi) \to [\ \varphi\ ,\ \psi\ ] \\
\textsf{CONS} : \Pi\!:\ (\varphi \to \psi) \to [\ \varphi\ ,\ \psi\ ] \\
\textsf{SKIP} : [\ \varphi\ ,\ \varphi\ ]
\end{array}
$$

Hoare Logic

$$\{\varphi\}\ P\ \{\psi\} \equiv P : [\ \varphi\ ,\ \psi\ ]$$

$$\begin{aligned}
&\mathsf{data}\ [_,_] : \mathsf{Assertion} \to \mathsf{Assertion} \to \mathsf{Set}_1\ \mathsf{where}\\
&\quad \mathsf{SEQ} : [\ \varphi\ ,\ \alpha\ ] \to [\ \alpha\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ]\\
&\quad \mathsf{CHO} : [\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ] \to [\ \varphi\ ,\ \psi\ ]\\
&\quad \mathsf{STAR} : [\ \varphi\ ,\ \varphi\ ] \to [\ \varphi\ ,\ \varphi\ ]\\
&\quad \mathsf{GUARD} : (g : \mathsf{Assertion}) \to [\ (g \to \varphi)\ ,\ \varphi\ ]\\
&\quad \mathsf{UPD} : (\Sigma{:}\ \varphi \to \Sigma{:}\ \psi) \to [\ \varphi\ ,\ \psi\ ]\\
&\mathsf{CONS} : \Pi{:}\ (\varphi \to \psi) \to [\ \varphi\ ,\ \psi\ ]\\
&\mathsf{SKIP} : [\ \varphi\ ,\ \varphi\ ]
\end{aligned}$$

See paper for relational semantics and soundness proof.

Deterministic Constructs

$$
\begin{aligned}
&\text{IFTHENELSE} : (g : \text{Assertion}) \\
&\qquad\qquad\quad \rightarrow [\,\varphi \times g\,,\,\psi\,] \\
&\qquad\qquad\quad \rightarrow [\,\varphi \times \neg\, g\,,\,\psi\,] \\
&\qquad\qquad\quad \rightarrow [\,\varphi\,,\,\psi\,] \\
&\text{IFTHENELSE}\ g\ P\ Q \\
&\quad = \text{CHO}\ (\text{SEQ}\ (\text{SEQ}\ (\text{CONS}\ \_,\_)\ (\text{GUARD}\ g))\ P) \\
&\qquad\qquad\ (\text{SEQ}\ (\text{SEQ}\ (\text{CONS}\ \_,\_)\ (\text{GUARD}\ (\neg\, g)))\ Q)
\end{aligned}
$$

Deterministic Constructs

$$
\begin{aligned}
\mathsf{IFTHENELSE} :\ & (g : \mathsf{Assertion}) \\
& \to [\ \varphi \times g\ ,\ \psi\ ] \\
& \to [\ \varphi \times \neg\ g\ ,\ \psi\ ] \\
& \to [\ \varphi\ ,\ \psi\ ]
\end{aligned}
$$

$$
\begin{aligned}
& \mathsf{IFTHENELSE}\ g\ P\ Q \\
& \quad = \mathsf{CHO}\ (\mathsf{SEQ}\ (\mathsf{SEQ}\ (\mathsf{CONS}\ \_,\_)\ (\mathsf{GUARD}\ g))\ P) \\
& \qquad\qquad (\mathsf{SEQ}\ (\mathsf{SEQ}\ (\mathsf{CONS}\ \_,\_)\ (\mathsf{GUARD}\ (\neg\ g)))\ Q)
\end{aligned}
$$

$$
\mathsf{WHILE} : (g : \mathsf{Assertion}) \to [\ g \times \varphi\ ,\ \varphi\ ] \to [\ \varphi\ ,\ \neg\ g \times \varphi\ ]
$$

$$
\begin{aligned}
\mathsf{WHILE}\ g\ P = \mathsf{SEQ}\ (\mathsf{STAR}\ (\mathsf{SEQ}\ (\mathsf{SEQ}\ & (\mathsf{CONS}\ (\mathsf{flip}\ \_,\_)) \\
& (\mathsf{GUARD}\ g)) \\
P))\quad & \\
(\mathsf{SEQ}\ (\mathsf{CONS}\ (\mathsf{flip}\ \_,\_))\quad & \\
(\mathsf{GUARD}\ (\neg\ g)))\quad &
\end{aligned}
$$

Ugly

This technique is very powerful!

Ugly

This technique is very powerful! **but...**

```
ls : OList 1 5
ls = Cons 1 (s≤s z≤n)
       (Cons 2 (s≤s z≤n)
         (Cons 3 (s≤s (s≤s z≤n))
           (Cons 4 (s≤s (s≤s (s≤s z≤n)))
             (Cons 5 (s≤s (s≤s (s≤s (s≤s z≤n))))
               (Nil (s≤s (s≤s (s≤s (s≤s (s≤s z≤n))))))))))
```

Uglier

```
ex : BST 2 10
ex = Branch 3 (Branch 2 (Leaf (s≤s (s≤s z≤n)))
     (Leaf (s≤s (s≤s z≤n))))
     (Branch 5 (Branch 4 (Leaf (s≤s (s≤s (s≤s z≤n))))
     (Leaf (s≤s (s≤s (s≤s (s≤s z≤n))))))
     (Branch 10 (Leaf (s≤s (s≤s (s≤s (s≤s (s≤s z≤n))))))
     (Leaf (s≤s (s≤s (s≤s (s≤s (s≤s (s≤s (s≤s (s≤s (s≤s (s≤s z≤n))))))))))
```

Ugliest

```
p : [ ⊤ , r ≡ sum a ]
p = SEQ (UPD (λ { ( σ , p ) → ( record σ { i = 0 ; r = 0 } , refl , z≤n)}))
      (SEQ (WHILE {r ≡ sum (take i a) × i ≤ length a} (i < length a)
              (UPD λ { ( σ , x , r≡sumᵢ , i≤n) →
                ( record σ { r = r ⦃ σ ⦄ + (a ⦃ σ ⦄ ! x) ; i = suc (i ⦃ σ ⦄)
                , trans (cong (_+ _) r≡sumᵢ) (take-i-plus-next x) , x ) }))
              (CONS λ { (¬i<len , r≡sumᵢ , i≤len) →
                trans r≡sumᵢ (trans (cong (λ h → sum (take h a))
                                          (not-<-but-≤ ¬i<len i≤len))
                                    (cong sum (take-length {ℓ = a}))) }))
```

Our Ideal

$$
\begin{array}{l}
\{\top\} \\
i := 0 \\
r := 0 \\
\{r = \Sigma_{k=0}^{i} a[k] \wedge i \leq \text{length } a\} \\
\textbf{while } i < \text{length } a \textbf{ do} \\
\quad r := r + A[i] \\
\quad i := i + 1 \\
\quad \{r = \Sigma_{k=0}^{i} a[k] \wedge i \leq \text{length } a\} \\
\textbf{od} \\
\{r = \Sigma_{k=0}^{\text{length } a} a[k]\}
\end{array}
$$

Our Ideal

$$
\begin{array}{l}
\{\top\} \\
i := 0 \\
r := 0 \\
\{r = \Sigma_{k=0}^{i} a[k] \wedge i \leq \text{length } a\} \\
\textbf{while } i < \text{length } a \textbf{ do} \\
\quad r := r + A[i] \\
\quad i := i + 1 \\
\quad \{r = \Sigma_{k=0}^{i} a[k] \wedge i \leq \text{length } a\} \\
\textbf{od} \\
\{r = \Sigma_{k=0}^{\text{length } a} a[k]\}
\end{array}
$$

Each assertion generates a proof obligation for **later** proof.

Our Ideal

$$
\begin{array}{l}
\{\top\} \\
i := 0 \\
r := 0 \\
\{r = \Sigma_{k=0}^{i} a[k] \land i \leq \text{length } a\} \\
\textbf{while } i < \text{length } a \textbf{ do} \\
\quad r := r + A[i] \\
\quad i := i + 1 \\
\quad \{r = \Sigma_{k=0}^{i} a[k] \land i \leq \text{length } a\} \\
\textbf{od} \\
\{r = \Sigma_{k=0}^{\text{length } a} a[k]\}
\end{array}
$$

Each assertion generates a proof obligation for **later** proof.

Separate **structure** from **proof**!

The Proof Delay Applicative

```
record Delay (X : Set ℓ) : Set (Level.suc ℓ) where
  constructor Prf
  field
    goals : List Set
    prove : HList goals → X
```

The Proof Delay Applicative

```
record Delay (X : Set ℓ) : Set (Level.suc ℓ) where
  constructor Prf
  field
    goals : List Set
    prove : HList goals → X

pure : X → Delay X
pure x = Prf [] (const x)
```

The Proof Delay Applicative

```
record Delay (X : Set ℓ) : Set (Level.suc ℓ) where
  constructor Prf
  field
    goals : List Set
    prove : HList goals → X

pure : X → Delay X
pure x = Prf [] (const x)

later : ∀{X} → Delay X
later {X} = Prf (X :: []) λ { (x :: []) → x }
```

The Proof Delay Applicative

```
record Delay (X : Set ℓ) : Set (Level.suc ℓ) where
  constructor Prf
  field
    goals : List Set
    prove : HList goals → X

pure : X → Delay X
pure x = Prf [] (const x)

later : ∀{X} → Delay X
later {X} = Prf (X :: []) λ { (x :: []) → x }

_⊛_ : Delay (A → B) → Delay A → Delay B
Prf goals₁ prove₁ ⊛ Prf goals₂ prove₂
  = Prf (goals₁ ++ goals₂)
        λ hl → prove₁ (takeH hl) (prove₂ (dropH hl))
```

Deferring the Details

$$\mathsf{nil} : \mathsf{Delay}\ (\mathsf{OList}\ m\ n)$$
$$\mathsf{nil} = (\!|\ \mathsf{Nil}\ \mathsf{later}\ |\!)$$

$$\_\mathsf{cons}\_ : (x : \mathbb{N}) \to \mathsf{Delay}\ (\mathsf{OList}\ x\ n) \to \mathsf{Delay}\ (\mathsf{OList}\ m\ n)$$
$$x\ \mathsf{cons}\ xs = (\!|\ (\mathsf{Cons}\ x)\ \mathsf{later}\ xs\ |\!)$$

Deferring the Details

$$\mathsf{nil} : \mathsf{Delay}\ (\mathsf{OList}\ m\ n)$$
$$\mathsf{nil} = (\!|\ \mathsf{Nil}\ \mathsf{later}\ |\!)$$

$$\_\mathsf{cons}\_ : (x : \mathbb{N}) \to \mathsf{Delay}\ (\mathsf{OList}\ x\ n) \to \mathsf{Delay}\ (\mathsf{OList}\ m\ n)$$
$$x\ \mathsf{cons}\ xs = (\!|\ (\mathsf{Cons}\ x)\ \mathsf{later}\ xs\ |\!)$$

```
example : OList 1 5
example = structure: 1 cons 2 cons 3 cons 4 cons 5 cons nil
          proofs:
             s≤s z≤n
          :: s≤s z≤n
          :: s≤s (s≤s z≤n)
          :: s≤s (s≤s (s≤s z≤n))
          :: s≤s (s≤s (s≤s (s≤s z≤n)))
          :: s≤s (s≤s (s≤s (s≤s (s≤s z≤n))))
          :: []
          done
```

Deferring for Trees

$$
\begin{array}{l}
\mathsf{leaf} : \mathsf{Delay}\ (\mathsf{BST}\ m\ n) \\
\mathsf{leaf} = (\!|\ \mathsf{Leaf}\ \mathsf{later}\ |\!)
\end{array}
$$

$$
\begin{array}{l}
\mathsf{branch} : (x : \mathbb{N}) \to \mathsf{Delay}\ (\mathsf{BST}\ m\ x) \to \mathsf{Delay}\ (\mathsf{BST}\ x\ n) \\
\qquad\quad \to \mathsf{Delay}\ (\mathsf{BST}\ m\ n) \\
\mathsf{branch}\ x\ l\ r = (\!|\ (\mathsf{Branch}\ x)\ l\ r\ |\!)
\end{array}
$$

$$
\begin{array}{l}
\mathsf{example}_2 : \mathsf{BST}\ 2\ 10 \\
\mathsf{example}_2 = \mathsf{structure{:}}\ \mathsf{branch}\ 3 \\
\qquad\qquad\qquad\qquad (\mathsf{branch}\ 2\ \mathsf{leaf}\ \mathsf{leaf}) \\
\qquad\qquad\qquad\qquad (\mathsf{branch}\ 5 \\
\qquad\qquad\qquad\qquad\quad (\mathsf{branch}\ 4\ \mathsf{leaf}\ \mathsf{leaf}) \\
\qquad\qquad\qquad\qquad\quad (\mathsf{branch}\ 10\ \mathsf{leaf}\ \mathsf{leaf})) \\
\qquad\qquad\quad \mathsf{proofs{:}}\ \langle \textit{omitted for brevity} \rangle \\
\qquad\qquad\quad \mathsf{done}
\end{array}
$$

TRIP

$$\mathsf{assert} : (\varphi : \mathsf{Assertion}) \to \mathsf{Delay}\ [\ \varphi\ ,\ \psi\ ]$$
$$\mathsf{assert}\ \varphi = (\!|\ \mathsf{CONS\ later}\ |\!)$$

$$P\ ;\ Q = (\!|\ \mathsf{SEQ}\ P\ Q\ |\!)$$

$$\mathsf{if}\ g\ \mathsf{then}\ p\ \mathsf{else}\ q\ \mathsf{fi} = (\!|\ (\mathsf{IFTHENELSE}\ g)\ p\ q\ |\!)$$

$$\mathsf{while}\ g\ \mathsf{begin}\ P\ \mathsf{end} = (\!|\ (\mathsf{WHILE}\ g)\ P\ |\!)$$

$$\mathsf{upd}\ u = \mathsf{pure}\ (\mathsf{UPD}\ u)$$

Swap and Assignment Macros

```
record SwapState : Set where
  field
    i : ℕ
    j : ℕ
    temp : ℕ
```

Swap and Assignment Macros

```
record SwapState : Set where
  field
    i : ℕ
    j : ℕ
    temp : ℕ
```

swp : $\forall \{I\ J : \mathbb{N}\} \rightarrow [\ \mathsf{i} \equiv I \times \mathsf{j} \equiv J\ ,\ \mathsf{j} \equiv I \times \mathsf{i} \equiv J\ ]$

Swap and Assignment Macros

```
record SwapState : Set where
  field
    i : ℕ
    j : ℕ
    temp : ℕ

swp : ∀{I J : ℕ} → [ i ≡ I × j ≡ J , j ≡ I × i ≡ J ]

swp = structure:
        upd (λ { (σ , p) → record σ { temp = i ⦃ σ ⦄ } , p }) ;
        upd (λ { (σ , p) → record σ { i = j       ⦃ σ ⦄ } , p }) ;
        upd (λ { (σ , p) → record σ { j = temp ⦃ σ ⦄ } , p })
        proofs: []
        done
```

Swap and Assignment Macros

```
record SwapState : Set where
  field
    i : ℕ
    j : ℕ
    temp : ℕ
```

$$\boxed{\{\varphi[{}^{e}/_{x}]\}\ x := e\ \{\varphi\}}$$

```
swp : ∀{I J : ℕ} → [ i ≡ I × j ≡ J , j ≡ I × i ≡ J ]

swp = structure:
        upd (λ { (σ , p) → record σ { temp = i ⦃ σ ⦄ } , p }) ;
        upd (λ { (σ , p) → record σ { i = j       ⦃ σ ⦄ } , p }) ;
        upd (λ { (σ , p) → record σ { j = temp ⦃ σ ⦄ } , p })
        proofs: []
        done
```

Swap and Assignment Macros

```
record SwapState : Set where
  field
    i : ℕ
    j : ℕ
    temp : ℕ
```

$$\boxed{\{\varphi[^e/_x]\}\ x := e\ \{\varphi\}}$$

s

s

```
swp′ : ∀{I J : ℕ} → [ i ≡ I × j ≡ J , j ≡ I × i ≡ J ]
swp′ = structure:
           temp := i ; i := j ; j := temp
       proofs: []
       done
```

Sum and Guarded Assignment

```
record SumState : Set where
  field
    i : ℕ
    arr : List ℕ
    total : ℕ
```

Sum and Guarded Assignment

```
record SumState : Set where
  field
    i : ℕ
    arr : List ℕ
    total : ℕ
```

Ideally, we would write:

```
i := 0 ;
total := 0 ;
while (i < length arr) begin
  total := (total + arr !! i) ;
  i := (1 + i)
end
```

Sum and Guarded Assignment

```
record SumState : Set where
  field
    i : ℕ
    arr : List ℕ
    total : ℕ
```

Ideally, we would write:

```
i := 0 ;
total := 0 ;
while (i < length arr) begin
  total := (total + arr !! i) ;
  i := (1 + i)
end
```

$\_!!\_ \; : \; \mathsf{List}\; A \to \mathbb{N} \to A$

Sum and Guarded Assignment

$$
\begin{aligned}
&\_!\_ : \forall \{n\} \to (ls : \mathsf{List}\ A) \to n < \mathsf{length}\ ls \to A \\
&\_!\_\ \{n = \mathsf{suc}\ \_\}\ (x :: ls)\ (\mathsf{s{\le}s}\ n) = ls\ !\ n \\
&\_!\_\ \{n = \mathsf{zero}\ \}\ (x :: ls)\ (\mathsf{s{\le}s}\ n) = x
\end{aligned}
$$

Ideally, we would write:

$$\boxed{\_!!\_ : \mathsf{List}\ A \to \mathbb{N} \to A}$$

$$
\begin{aligned}
&\mathsf{i} := 0\ ; \\
&\mathsf{total} := 0\ ; \\
&\mathbf{while}\ (\mathsf{i} < \mathsf{length}\ \mathsf{arr})\ \mathbf{begin} \\
&\quad \boxed{\mathsf{total} := \mathsf{total} + (\mathsf{arr}\ !\ i{<}len)\ \mathsf{given}\ i{<}len : (\mathsf{i} < \mathsf{length}\ \mathsf{arr})} \\
&\quad \mathsf{i} := (1 + \mathsf{i}) \\
&\mathbf{end}
\end{aligned}
$$

Sum

```
lsum : [ ⊤ , result ≡ sum arr ]
lsum =

  structure:
    assert ⊤ ;
    i := 0 ;
    total := 0 ;
    assert (i ≡ 0 × total ≡ 0) ;
    while (i < length arr) begin
      total := total + (arr ! i<len)
        given i<len : (i < length arr) ;
      i := (1 + i)
    end ;
    assert (¬ i < length arr
          × total ≡ sum (take i arr)
          × i ≤ length arr )
```

In the paper...

- A proof delay monad that requires replicating structure in the proofs.

In the paper...

- A proof delay monad that requires replicating structure in the proofs.
- Semantics and proofs about Trip.
- A lot of implementation details about the macros.

In the paper...

- A proof delay monad that requires replicating structure in the proofs.
- Semantics and proofs about Trip.
- A lot of implementation details about the macros.
- A more involved example (cumulative sum).

In the paper...

- A proof delay monad that requires replicating structure in the proofs.
- Semantics and proofs about Trip.
- A lot of implementation details about the macros.
- A more involved example (cumulative sum).
- Comparisons to ornaments, Coq's `Program`, Morgan's refinement calculus, Schirmer's SIMPL and more!

In the paper...

- A proof delay monad that requires replicating structure in the proofs.
- Semantics and proofs about Trip.
- A lot of implementation details about the macros.
- A more involved example (cumulative sum).
- Comparisons to ornaments, Coq's `Program`, Morgan's refinement calculus, Schirmer's SIMPL and more!

Future work:

- A language importer.
- Integrating proof automation.
- State management, recursion, procedures.
- Other uses for the proof delay idea?

Thank you!

All the code, examples and proofs can be found here!

`http://www.github.com/liamoc/dddp`