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:

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

Dependently-typed Data Structures

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

```
data OList (m n : ℕ) : Set where
  Nil : (m ≤ n) → OList m n
  Cons : (x : ℕ) → (m ≤ x) → OList x n → OList m n
```

```
data BST (m n : ℕ) : Set where
  Leaf : (m ≤ n) → BST m n
  Branch : (x : ℕ) → BST m x → BST x n → BST m n
```

Hoare Logic

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

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

Hoare Logic

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

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

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

Hoare Logic

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

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

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

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

Hoare Logic

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

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

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

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

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

Hoare Logic

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

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

Hoare Logic

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

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

Hoare Logic

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

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

Hoare Logic

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

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

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 ]
CONS : Π: (φ → ψ) → [ φ , ψ ]
SKIP : [ φ , φ ]
```

See paper for relational semantics and soundness proof.

Deterministic Constructs

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

Deterministic Constructs

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

$$\begin{aligned}
&\mathsf{WHILE} : (g : \mathsf{Assertion}) \to [\ g \times \varphi\ ,\ \varphi\ ] \to [\ \varphi\ ,\ \neg\, g \times \varphi\ ] \\
&\mathsf{WHILE}\ g\ P = \mathsf{SEQ}\ (\mathsf{STAR}\ (\mathsf{SEQ}\ (\mathsf{SEQ}\ (\mathsf{CONS}\ (\mathsf{flip}\ _,_)) \\
&\qquad\qquad (\mathsf{GUARD}\ g)) \\
&\qquad\qquad P)) \\
&\qquad (\mathsf{SEQ}\ (\mathsf{CONS}\ (\mathsf{flip}\ _,_)) \\
&\qquad\qquad (\mathsf{GUARD}\ (\neg\, g)))
\end{aligned}$$

Ugly

```
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 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≡sum_i , i≤n) →
                      ( record σ { r = r {| σ |} + (a {| σ |} ! x) ; i = suc (i {| σ |})
                      , trans (cong (_+ _) r≡sum_i) (take-i-plus-next x) , x ) }))
                    (CONS λ { (¬i<len , r≡sum_i , i≤len) →
                      trans r≡sum_i (trans (cong (λ h → sum (take h a))
                                                 (not-<-but-≤ ¬i<len i≤len))
                                           (cong sum (take-length {ℓ = a}))) }))
```

Our Ideal

$\{\top\}$
$i := 0$
$r := 0$
$\{r = \Sigma_{k=0}^{i} a[k] \wedge i \leq \text{length } a\}$
while $i < \text{length } a$ **do**
  $r := r + A[i]$
  $i := i + 1$
  $\{r = \Sigma_{k=0}^{i} a[k] \wedge i \leq \text{length } a\}$
od
$\{r = \Sigma_{k=0}^{\text{length } a} a[k]\}$

Our Ideal

$\{\top\}$
$i := 0$
$r := 0$
$\{r = \Sigma_{k=0}^{i} a[k] \wedge i \leq \text{length } a\}$
while $i < \text{length } a$ **do**
  $r := r + A[i]$
  $i := i + 1$
  $\{r = \Sigma_{k=0}^{i} a[k] \wedge i \leq \text{length } a\}$
od
$\{r = \Sigma_{k=0}^{\text{length } a} a[k]\}$

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

Our Ideal

$\{\top\}$
$i := 0$
$r := 0$
$\{r = \Sigma_{k=0}^{i} a[k] \wedge i \leq \text{length } a\}$
while $i < \text{length } a$ **do**
  $r := r + A[i]$
  $i := i + 1$
  $\{r = \Sigma_{k=0}^{i} a[k] \wedge i \leq \text{length } a\}$
od
$\{r = \Sigma_{k=0}^{\text{length } a} a[k]\}$

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

```
nil : Delay (OList m n)
nil = (| Nil later |)

_cons_ : (x : ℕ) → Delay (OList x n) → Delay (OList m n)
x cons xs = (| (Cons x) 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

```
leaf : Delay (BST m n)
leaf = (| Leaf later |)

branch : (x : ℕ) → Delay (BST m x) → Delay (BST x n)
         → Delay (BST m n)
branch x l r = (| (Branch x) l r |)


example₂ : BST 2 10
example₂ = structure: branch 3
                        (branch 2 leaf leaf)
                        (branch 5
                          (branch 4 leaf leaf)
                          (branch 10 leaf leaf))
           proofs: ⟨omitted for brevity⟩
           done
```

Trip

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

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

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

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

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

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

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
```

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

```
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
```

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

```
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:

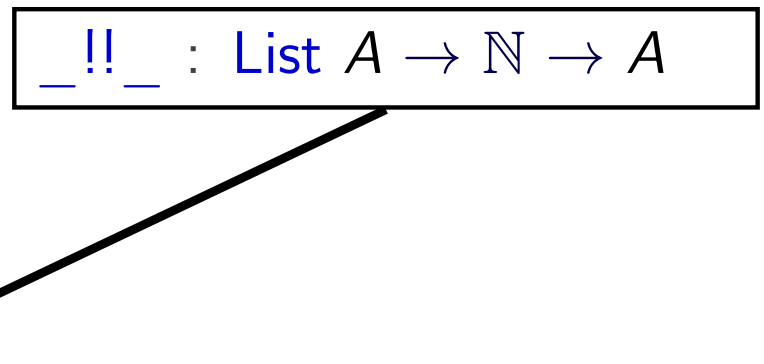
```
i := 0 ;
total := 0 ;
while (i < length arr) begin
  total := (total + arr !! i) ;
  i := (1 + i)
end
```

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
```



Sum and Guarded Assignment

$$\_!\_ : \forall \{n\} \to (ls : \text{List } A) \to n < \text{length } ls \to A$$
$$\_!\_ \{n = \text{suc } \_\} (x :: ls) (s{\leq}s\ n) = ls\ !\ n$$
$$\_!\_ \{n = \text{zero } \} (x :: ls) (s{\leq}s\ n) = x$$

Ideally, we would write:

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

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

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`