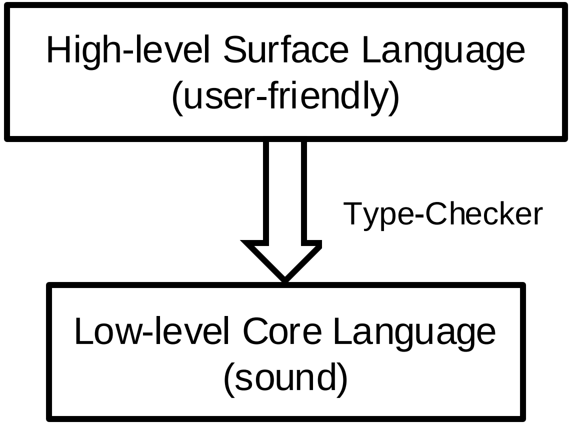
A generic translation from case trees to eliminators

Kayleigh Lieverse, Lucas Escot, Jesper Cockx

Dependently Typed Languages

- Program + Proof
- Limitation: errors in elaboration



Pattern Matching and Eliminators

```
data ℕ : Set where
    zero : ℕ
    suc : ℕ → ℕ

_+_ : ℕ → ℕ → ℕ
zero + m = m
suc n + m = suc (n + m)
```

```
ℕ-elim : (P : ℕ → Set)
     (mZero : P zero)
     (mSuc : (n : ℕ) → (h : P n) → P (suc n))
     (n : ℕ) → P n
ℕ-elim P mZero mSuc zero = mZero
ℕ-elim P mZero mSuc (suc n) = mSuc n (ℕ-elim P mZero mSuc n)
```

```
_+'_ : ℕ → ℕ → ℕ
_+'_ n m = ℕ-elim (λ n → ℕ → ℕ)
            (λ m → m)
            (λ n h m → suc (h m))
            n m
```

TUDelft

Dependent Pattern Matching and Eliminators

```
data Vec (A : Set) : ℕ → Set where
  nil : Vec A zero
  cons : (n : ℕ) → A → Vec A n → Vec A (suc n)
```

```
head : {A : Set} (h : ℕ) (v : Vec A (suc h)) → A
head h (cons .h a xs) = a
```

```
NoConfusion : ℕ → ℕ → Set
```

```
basic-analysis : {A : Set}
    → (mNil : (h : ℕ) (v : Vec A (suc h))
      → (zero , nil {A = A}) ≡ (suc h , v) → A)

  P : (n : ℕ) (xs : Vec A n) → Set
  P n xs = (h : ℕ) (v : Vec A (suc h)) → (_,_ {B = Vec A} n xs) ≡ (suc h , v) → A
```

```
Vec-case : {A : Set} (P : (n : ℕ) (xs : Vec A n) → Set)
    → (mNil : P zero nil)
    → (mCons : (n : ℕ) (a : A) (xs : Vec A n) → P (suc n) (cons n a xs))
    → (n : ℕ) (xs : Vec A n) → P n xs
Vec-case P mNil mCons n xs = Vec-elim P mNil (λ n a xs h → mCons n a xs) n xs
```

From Pattern Matching to Case Trees

```
data ℕ : Set where
    zero : ℕ
    suc : ℕ → ℕ

_+_ : ℕ → ℕ → ℕ
zero + m = m
suc n + m = suc (n + m)
```

$$[(n : \mathbb{N})(m : \mathbb{N})]\ \underline{n}\ m \begin{cases} [(m : \mathbb{N})]\ \mathtt{zero}\ m \mapsto m \\ [(n : \mathbb{N})(m : \mathbb{N})]\ (\mathtt{suc}\ n)\ m \mapsto \mathtt{suc}\ (n + m) \end{cases}$$

From Case Trees to Eliminators

- Generic Representation Case Tree
- Unification Algorithm
- Evaluation Function
- Discussion

Generic Representation Case Tree

- Case Trees Represent Functions

```
tail : (n : ℕ) → Vec A (suc n) → Vec A n
tail n (cons n x xs) = xs
```

Generic Representation Case Tree: Telescopes

- Telescope:

```
data Telescope : ℕ → Set₁ where
    nil : Telescope 0
    cons : (S : Set) (E : S → Telescope n) → Telescope (suc n)
```

- Interpretation:

```
⟦_⟧telD : (Δ : Telescope n) → Set
⟦ nil        ⟧telD = ⊤
⟦ cons S E ⟧telD = Σ[ s ∈ S ] ⟦ E s ⟧telD
```

TUDelft

Generic Representation Case Tree: Telescopes

- Case Tree Type:

```
data CaseTree (Δ : Telescope n)(T : ⟦ Δ ⟧telD → Set ℓ) : Set (lsuc ℓ)
```

```
tail : (n : ℕ) → Vec A (suc n) → Vec A n
tail n (cons n x xs) = xs

CTTail : CaseTree (n ∈ ℕ , xs ∈ Vec A (suc n) , nil) (λ { (n , xs , tt) → Vec A n })
```

Generic Representation Case Trees: Data Types

- Number of constructors (arguments)
- Universe of data type descriptions

```
data Vec (A : Set) : ℕ → Set where
  nil : Vec A zero
  cons : (n : ℕ) → A → Vec A n → Vec A (suc n)
```

```
data ConDesc (is : Telescope iₙ) : ℕ → Set₁ where
  one' : ⟦ is ⟧telD → ConDesc is 0
  Σ'   : (S : Set) → (D : S → ConDesc is aₙ) → ConDesc is (suc aₙ)
  ×'   : ⟦ is ⟧telD → ConDesc is aₙ → ConDesc is (suc aₙ)

DataDesc : Telescope iₙ → ℕ → Set₁
DataDesc is cₙ = Fin cₙ → Σ ℕ (ConDesc is)
```

TUDelft

Unification

```
head : (n : ℕ) → Vec A (suc n) → A
head n (cons n x xs) = x

tail : (n : ℕ) → Vec A (suc n) → Vec A n
tail n (cons n x xs) = xs
```

- Constructor nil: $(n : \mathbb{N})(\mathsf{suc}\ n \overset{?}{=} \mathsf{zero})$
- Constructor cons: $(n\ m : \mathbb{N})(\mathsf{suc}\ n \overset{?}{=} \mathsf{suc}\ m)$

Unification

$$\texttt{solution}:(x:A)(e:x\equiv_A t)\simeq ()$$

$$\texttt{deletion}:(e:t\equiv_A t)\simeq ()$$

$$\texttt{injectivity}_{\mathtt{c}}:(\mathtt{c}\ \bar{s}\equiv_{\mathtt{D}}\mathtt{c}\ \bar{t})\simeq(\bar{s}\equiv_{\Delta_{\mathtt{c}}}\bar{t})$$

$$(\mathtt{suc}\ n\equiv_{\mathbb{N}}\mathtt{suc}\ m)\simeq(n\equiv_{\mathbb{N}} m)$$

$$\texttt{conflict}_{\mathtt{c}_1,\mathtt{c}_2}:(\mathtt{c}_1\ \bar{s}\equiv_{\mathtt{D}}\mathtt{c}_2\ \bar{t})\simeq\bot$$

$$(n:\mathbb{N})(\mathtt{zero}\equiv_{\mathbb{N}}\mathtt{suc}\ n)\simeq\bot$$

Unification Algorithm

```
unifyTel : {Δ : Telescope n} (u : Unification Δ) → Σ ℕ Telescope

unify    : {Δ : Telescope n} (u : Unification Δ) → ⟦ Δ ⟧telD → ⟦ proj₂ (unifyTel u) ⟧telD
unify'   : {Δ : Telescope n} (u : Unification Δ) → ⟦ proj₂ (unifyTel u) ⟧telD → ⟦ Δ ⟧telD
```

Evaluation Function

```
eval : {Δ : Telescope n}{T : ⟦ Δ ⟧telD → Set ℓ}
  (ct : CaseTree Δ T) (args : ⟦ Δ ⟧telD)
  → T args
```

- Leaf : clause of function
- Node: case split
  - Eliminate variable
  - Basic analysis
  - All telescope functions are section-retraction pairs

Discussion

- What is possible?
  - Course-of-value iteration
  - Higher-dimensional unification
- What is not possible?
  - Cycle rule (without a lot of extra work)
  - Not indexed data types
- Extending Agda?

Questions?

Generic Representation Case Tree

```
data CaseTree (Δ : Telescope n)(T : ⟦ Δ ⟧telD → Set ℓ) : Set (lsuc ℓ) where
    leaf : (t : (args : ⟦ Δ ⟧telD) → T args) → CaseTree Δ T
    node : {D : DataDesc is cₙ} (p : Δ [ k ]∶Σ[ ⟦ is ⟧telD ] (μ D))
      → (bs : (cᵢ : Fin cₙ)
        → Σ[ u ∈ Unification (expandTel Δ (conTel (μ D) (proj₂ (D cᵢ))) p
              (λ args → ⟨ cᵢ , telToCon args ⟩ ))]
            (CaseTree (proj₂ (unifyTel u)) (λ args → T (shrink p (unify' u args)))))
      → CaseTree Δ T
```

Evaluation Function

```
eval : {Δ : Telescope n}{T : ⟦ Δ ⟧telD → Set ℓ} →
  (ct : CaseTree Δ T) (args : ⟦ Δ ⟧telD)
  → T args
eval (leaf f) args = f args
eval {T = T} (node {is = is} {D = D} p bs) args
    = case-μ D (λ d' x' → (d' , x') ≡ (d , ret) → T args) cs d ret refl where

  d : ⟦ is ⟧telD
  d = proj₁ (args Σ[ p ])

  ret : μ D d
  ret = proj₂ (args Σ[ p ])

  cs : (d' : ⟦ is ⟧telD) (x : ⟦ D ⟧ (μ D) d') → (d' , ⟨ x ⟩) ≡ (d , ret) → T args
```

TUDelft