Polynomial Time and Dependent Types

Author: Bob Atkey
Presenter: Eric Bond
MPLS Reading Group
March 2024

Motivation

```
insertionSort :: Ord a => [a] -> [a]
insertionSort [] = []
insertionSort [x] = [x]
insertionSort (x:xs) = insert $ insertionSort xs
    where insert [] = [x]
          insert (y:ys)
             | x < y = x : y : ys
             | otherwise = y : insert ys
```

```
record SortingAlgorithm : Set (a ⊔ ℓ₁ ⊔ ℓ₂) where
  field
    sort   : List A → List A

    -- The output of `sort` is a permutation of the input
    sort-↭ : ∀ xs → sort xs ↭ xs

    -- The output of `sort` is sorted.
    sort-↗ : ∀ xs → Sorted (sort xs)
```

Existing Solution - RaML

```
let rec partition f l =
  match l with
    | [] -> ([],[])
    | x::xs ->
      let (cs,bs) = partition f xs in
      Raml.tick 1.0;
      if f x then
    (cs,x::bs)
      else
    (x::cs,bs)



let rec quicksort gt = function
  | [] -> []
  | x::xs ->
      let ys, zs = partition (gt x) xs in
      append (quicksort gt ys) (x :: (quicksort gt zs))
```

```
Analyzing function quicksort ...
== quicksort : ['a -> 'a -> bool; 'a list] -> 'a list

  Non-zero annotations of the argument:
         1  <--  (*, [::(*); ::(*)])

  Non-zero annotations of result:
  Simplified bound:
     -0.5*M + 0.5*M^2
   where
     M is the number of ::-nodes of the 2nd component of the argument
--
  Mode:          upper
  Metric:        ticks
  Degree:        3
  Run time:      0.09 seconds
  #Constraints:  1635
```

https://www.raml.co/interface/

Existing Solution - CALF

```
sort/is-bounded : ∀ l → IsBoundedG (Σ⁺ (list A) (sorted-of l)) (sort l) (sort/cost l)
sort/is-bounded []       = ≤⁻-refl
sort/is-bounded (x ∷ xs) =
  let open ≤⁻-Reasoning cost in
  begin
    ( bind cost (sort xs) λ (xs' , xs↭xs' , sorted-xs') →
      bind cost (insert x xs' sorted-xs') λ _ →
      step⋆ zero
    )
  ≤⟨ bind-monoʳ-≤⁻ (sort xs) (λ (xs' , xs↭xs' , sorted-xs') → insert/is-bounded x xs' sorted-xs') ⟩
    ( bind cost (sort xs) λ (xs' , xs↭xs' , sorted-xs') →
      step⋆ (length xs')
    )
  ≡˘⟨
    Eq.cong
      (bind cost (sort xs))
      (funext λ (xs' , xs↭xs' , sorted-xs') → Eq.cong step⋆ (↭-length xs↭xs'))
  ⟩
    ( bind cost (sort xs) λ _ →
      step⋆ (length xs)
    )
  ≤⟨ bind-monoˡ-≤⁻ (λ _ → step⋆ (length xs)) (sort/is-bounded xs) ⟩
    step⋆ ((length xs ²) + length xs)
  ≤⟨ step⋆-mono-≤⁻ (N.+-mono-≤ (N.*-monoʳ-≤ (length xs) (N.n≤1+n (length xs))) (N.n≤1+n (length xs))) ⟩
    step⋆ (length xs * length (x ∷ xs) + length (x ∷ xs))
  ≡⟨ Eq.cong step⋆ (N.+-comm (length xs * length (x ∷ xs)) (length (x ∷ xs))) ⟩
    step⋆ (length (x ∷ xs) ²)
  ≡⟨⟩
    sort/cost (x ∷ xs)
  ∎
```

```
a length , sort ∈𝒪(λ n → n ²)
ed
```

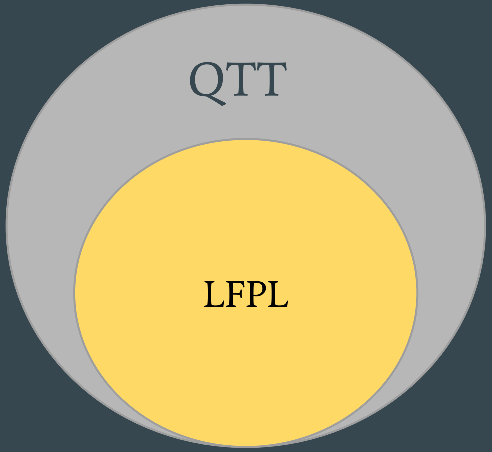
Shift in Perspective

Proving complexity bounds in an unrestricted language

verses

Programming in a language which guarantees complexity bounds

Overview



+ Applications

Implicit Complexity Theory

Complexity Class	Type Theory
PTIME/FP	[Hof03] [DLH09]
PSPACE	[Hof03]
2k-EXP/2k-FEXP	[BG20]
P/Poly	[MT15]
LOGSPACE	[Maz15] [ADLV22b]
PP	[DLKO21]
P#	[DLKO22]
EQP, BQP, ZQP	[DMZ10]

Some ICC Results

Implicit Complexity Theory

Linear types and non-size-increasing polynomial time computation

Martin Hofmann

Ludwig-Maximilians-Universität München, Theoretical Computer Science, Oettingenstrasse 67, 80538 München, Germany

Received 27 January 2000; revised 19 January 2001

Abstract

We propose a linear type system with recursion operators for inductive datatypes which ensures that all definable functions are polynomial time computable. The system improves upon previous such systems in that recursive definitions can be arbitrarily nested; in particular, no predicativity or modality restrictions are made.

Keywords: Complexity theory; Type system; Linear types; Higher-order function; Resources

LFPL

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \multimap B} \qquad \frac{\Gamma_1 \vdash M : A \multimap B \qquad \Gamma_2 \vdash N : A}{\Gamma_1, \Gamma_2 \vdash M\,N : B}$$

$$\frac{\Gamma_1 \vdash M : A \qquad \Gamma_2 \vdash N : B}{\Gamma_1, \Gamma_2 \vdash (M, N) : A \otimes B} \qquad \frac{\Gamma_1 \vdash M : A \otimes B \qquad \Gamma_2, x : A, y : B \vdash N : C}{\Gamma_1, \Gamma_2 \vdash \mathrm{let}\,(x, y) = M \,\mathrm{in}\, N : C}$$

$$\frac{\Gamma \vdash M : \diamond}{\Gamma \vdash \mathrm{zero}(M) : \mathrm{Nat}} \qquad \frac{\Gamma_1 \vdash M : \diamond \qquad \Gamma_2 \vdash N : \mathrm{Nat}}{\Gamma_1, \Gamma_2 \vdash \mathrm{succ}(M, N) : \mathrm{Nat}}$$

$$\frac{d : \diamond \vdash M_z : A \qquad d : \diamond, x : A \vdash M_s : A \qquad \Gamma \vdash N : \mathrm{Nat}}{\Gamma \vdash \mathrm{rec}\, N\, \{\mathrm{zero}(d) \mapsto M_z; \mathrm{succ}(d, x) \mapsto M_s\} : A}$$

LFPL

```
let
    concat: int list list -> int list =
        fn xss: int list list =>
        iter xss {
            [] => []: int
          | cons(d, xs, _) with y =>
                iter xs {
                    [] => y
                  | cons(d', x, _) with y' =>
                        cons(d', x, y')
                }
        }
in
    concat
    -- Example input:
    -- ([[1,2,3]: int, [4,5,6]: int, [7,8,9]: int]: int list)
end
```

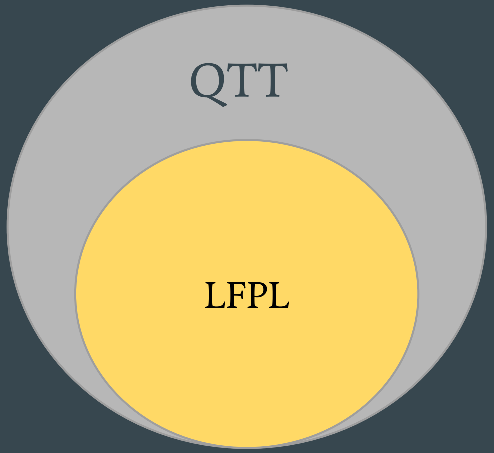
https://github.com/ishantheperson/LFPL/blob/main/test/concat.lfpl

QTT + LFPL Soundness Proof Statement

Theorem 6.2 (Soundness for the LFPL-style System). *If we have a term $n \overset{1}{:} \mathrm{Nat} \vdash M \overset{1}{:} T(n)$ then there exists a realising expression E and polynomial p such that for all $n \in \mathbb{N}$, there exists $v \in \mathcal{V}$ and $k \in \mathbb{N}$ such that $E, [\mathrm{natValue}(n)] \Downarrow_k v$, $k \leq p(n+1)$ and v is a realising value for $[\![M]\!](n) \in [\![T]\!](n)$.*

```
-- The polytime property for LFPL
poly-time : ∀ {X} →
   (f : ℕ ⊢ `nat ⇒ X) →
   Σ[ p ∈ Poly ] ∀ n →
      Σ[ v ∈ val ] Σ[ k ∈ ℕ ]
        f .realiser .expr 0 , (nil ,- nat-val n) ⇓[ k ] v
        × k ≤ ⟦ p ⟧ (suc n)
poly-time f .proj₁ = f .realiser .potential .proj₂
poly-time f .proj₂ n =
  r .result ,
  r .steps ,
  r .evaluation ,
  under-time
  where
  r = f .realises n nil (size (suc n)) (nat-val n) (refl , ≤-refl , (λ _ _ → ≤-refl))
```

Overview



Quantitative Type Theory

σ = 0 "normal" DTT
σ = 1 for LFPL typing rules

$$x_1 \overset{\rho_1}{:} S_1, \ldots, x_n \overset{\rho_n}{:} S_n \vdash M \overset{\sigma}{:} T$$

Qtt Typing Judgement

Q: Should the usage of variables in types cost anything?

A: NO

$$\text{insertionSortCorrect} : (xs \overset{\omega}{:} \text{IList}\ A) \to \text{Sorted}(xs, \text{insertionSort}\ \text{xs})$$

Quantitative Type Theory

$$x_1 \overset{\rho_1}{:} S_1, \ldots, x_n \overset{\rho_n}{:} S_n \vdash M \overset{\sigma}{:} T$$

Qtt Typing Judgement

$$\frac{\Gamma \vdash M \overset{1}{:} S}{0\Gamma \vdash M \overset{0}{:} S}$$

“LFPL” →
“DTT”

Quantitative TT + LFPL

- Pi and Sigma types
- Identity types
- Universe types
- Diamonds
- Booleans
- Natural numbers
- Lists
- Iterable lists
  - Constructed with diamonds
- ⭐Realizability types⭐

Diamonds (again)

$$\frac{\Gamma\ \mathrm{ctxt}}{0\Gamma \vdash \diamond\ \mathrm{type}}$$

Type Formation

$$\frac{\Gamma\ \mathrm{ctxt}}{0\Gamma \vdash * \overset{0}{:} \diamond}$$

Introduction

$$\frac{\Gamma \vdash M \overset{0}{:} \diamond}{\Gamma \vdash M \equiv * \overset{0}{:} \diamond}$$

Eta rule

Natural Numbers

$$\frac{\Gamma \vdash M \overset{\sigma}{:} \diamond}{\Gamma \vdash \mathrm{zero}(M) \overset{\sigma}{:} \mathrm{Nat}}$$

Zero constructor

$$\frac{\Gamma_1 \vdash M \overset{\sigma}{:} \diamond \qquad \Gamma_2 \vdash N \overset{\sigma}{:} \mathrm{Nat} \qquad 0\Gamma_1 = 0\Gamma_2}{\Gamma_1 + \Gamma_2 \vdash \mathrm{succ}(M, N) \overset{\sigma}{:} \mathrm{Nat}}$$

Succ constructor

$$\frac{\begin{array}{l} 0\Gamma, x \overset{0}{:} \mathrm{Nat} \vdash P \text{ type} \\ \Gamma \vdash M \overset{\sigma}{:} \mathrm{Nat} \\ 0\Gamma, d \overset{\sigma}{:} \diamond \vdash N_z \overset{\sigma}{:} P[\mathrm{zero}(*)/x] \\ 0\Gamma, d \overset{\sigma}{:} \diamond, n \overset{0}{:} \mathrm{Nat}, p \overset{\sigma}{:} P[n/x] \vdash N_s \overset{\sigma}{:} P[\mathrm{succ}(*, n)/x] \end{array}}{\Gamma \vdash \mathrm{rec}\, M \,\{\mathrm{zero}(d) \mapsto N_z; \mathrm{succ}(d, n; p) \mapsto N_s\} \overset{\sigma}{:} P[M/x]}$$

Recursor

Sigma

$$\frac{\Gamma \vdash M \overset{0}{:} (x \overset{\pi}{:} S) \otimes T}{\Gamma \vdash \mathrm{fst}(M) \overset{0}{:} S} \qquad \frac{\Gamma \vdash M \overset{0}{:} (x \overset{\pi}{:} S) \otimes T}{\Gamma \vdash \mathrm{snd}(M) \overset{0}{:} T[\mathrm{fst}(M)/x]}$$

Elimination when σ = 0

$$\frac{\Gamma_1 \vdash M \overset{\sigma}{:} (x \overset{\pi}{:} A) \otimes B \qquad \begin{matrix} 0\Gamma, z \overset{0}{:} (x \overset{\pi}{:} A) \otimes B \vdash C \\ \Gamma_2, x \overset{\sigma\pi}{:} A, y \overset{\sigma}{:} B \vdash N \overset{\sigma}{:} C[(x,y)/z] \end{matrix} \qquad 0\Gamma_1 = 0\Gamma_2}{\Gamma_1 + \Gamma_2 \vdash \mathrm{let}\ (x,y) = M\ \mathrm{in}\ N \overset{\sigma}{:} C[M/z]}$$

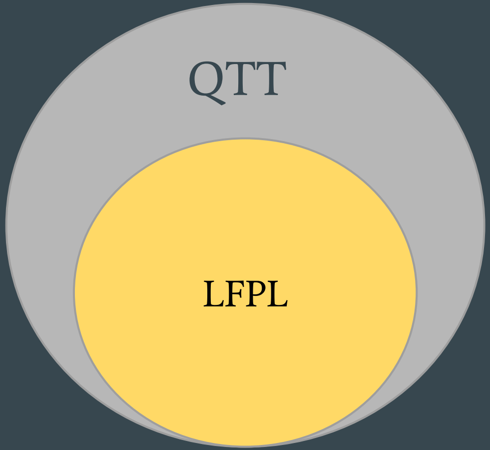
Elimination when σ = 0 or σ = 1

Constructing Data Types

$$\mathrm{IList}\,A = (n \overset{1}{:} \mathrm{Nat}) \otimes (\mathrm{rec}_{x.\mathsf{U}}\, n\, \{\mathrm{zero}(d) \mapsto I; \mathrm{succ}(d, n; p) \mapsto A \otimes p\})$$

$$\begin{array}{ll}
\mathrm{nil} \overset{1}{:} \diamond \to \mathrm{IList}\,A & \mathrm{cons} \overset{1}{:} \diamond \to A \to \mathrm{IList}\,A \to \mathrm{IList}\,A \\
\mathrm{nil}\, d = (\mathrm{zero}(d), *) & \mathrm{cons}\, d\, x\, xs = \mathrm{let}\, (n, \mathit{elems}) = xs\, \mathrm{in}\, (\mathrm{succ}(d, n), (x, \mathit{elems}))
\end{array}$$

Overview



\+ Applications

Realizability Types

$$\frac{0\Gamma \vdash A \text{ type}}{0\Gamma \vdash \mathbf{R}(A) \text{ type}}$$

Type Formation

$$\frac{0\Gamma \vdash M \overset{1}{:} A}{0\Gamma \vdash \mathbf{R}(M) \overset{\sigma}{:} \mathbf{R}(A)}$$

Introduction

$$\frac{\Gamma \vdash M \overset{\sigma}{:} \mathbf{R}(A)}{\Gamma \vdash \mathbf{R}^{-1}(M) \overset{\sigma'}{:} A}$$

Elimination

$$\mathbf{R}(\mathbf{R}^{-1}(M)) \equiv M$$

When σ = 0 or σ = 1

$$\mathbf{R}^{-1}(\mathbf{R}(M)) \equiv M$$

When σ = 0

Complexity Theory

$$\mathrm{PTIME}(A, P) = (f \overset{1}{:} \mathbf{R}(A \to \mathsf{Bool})) \otimes \left((a \overset{1}{:} A) \to (\mathbf{R}^{-1}(f)\ a = \mathsf{true}) \Leftrightarrow P\ a\right)$$

$$(A, P) \overset{\textsc{Poly}}{\Rightarrow} (B, Q) = (f \overset{1}{:} \mathbf{R}(A \to B)) \otimes \left((a \overset{1}{:} A) \to Q(\mathbf{R}^{-1}(f)\ a) \Leftrightarrow P\ a\right)$$

- "Proofs of PTIME(A, P), are carried out in the $\sigma = 0$ fragment, where we have the full power of Type Theory to aid us."
- "This definition is intrinsic, in the sense that, whichever of the polytime systems is chosen, proving that a decision problem is solvable in polytime is a **matter of programming**, without having to reason directly about machine models and step counting."
- "We have defined problems to have arbitrary types A as domains, rather than bitstrings, and so the notion of size attached to an input is intrinsic to the type A chosen."

Other Complexity Classes

$$
\begin{aligned}
&\textbf{data}\ \mathrm{ND}\ (A : \mathsf{U}) : \mathsf{U}\ \textbf{where} \\
&\quad \mathsf{return} : A \longrightarrow \mathrm{ND}\, A \\
&\quad \mathsf{choice} : (\mathrm{Bool} \longrightarrow \mathrm{ND}\, A) \longrightarrow \mathrm{ND}\, A
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{NP}(A, P) = {} & (f \overset{1}{:} \mathbf{R}(A \longrightarrow \mathrm{ND}(\mathrm{Bool}))) \otimes \\
& \Big((a \overset{1}{:} A) \longrightarrow \Big((bs \overset{1}{:} \mathrm{List}(\mathrm{Bool})) \otimes (\mathrm{runWithOracle}\ (\mathbf{R}^{-1}(f)\ a)\ bs = \mathrm{just\ true})\Big) \Leftrightarrow P\,a\Big)
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{data}\ \mathrm{Dist}\ (A : \mathsf{U}) : \mathsf{U}\ \textbf{where} \\
&\quad \mathsf{return} : A \longrightarrow \mathrm{Dist}\, A \\
&\quad \mathsf{choice} : \mathbb{Q}[0,1] \longrightarrow (\mathrm{Bool} \longrightarrow \mathrm{Dist}\, A) \longrightarrow \mathrm{Dist}\, A
\end{aligned}
$$

$$
\mathrm{BPP}(A, P) = \ (f \overset{1}{:} \mathbf{R}(A \to \mathrm{Dist}(\mathrm{Bool}))) \otimes \Big((a \overset{1}{:} A) \to (\mathrm{probTrue}\ (\mathbf{R}^{-1}(f)\ a) \geq \tfrac{2}{3}) \Leftrightarrow P\,a\Big)
$$

Concluding Remarks

- Capabilities
  - Write polytime programs
  - Prove their correctness
  - Define complexity theory concepts
- Future Work
  - Explicit resource bounds?
  - Internalize realizability proof (prove Cook Levin)?