Program Term Generation Through Enumeration of Indexed Data Types (Thesis Proposal)

Cas van der Rest

January 14, 2019

Contents

1 Introduction

What is the problem? Illustrate with an example. [1, 12]

What is/are your research questions/contributions? [3]

2 Background

What is the existing technology and literature that I'll be studying/using in my research [6, 10, 11, 14]

2.1 Dependently Typed Programming & Agda

2.1.1 Propositions as Types

2.1.2 Codata

2.2 Property Based Testing

2.2.1 Existing Libraries

2.2.2 Generating Test Data

2.3 Generic Programming & Type Universes

2.3.1 Regular Datatypes

2.3.2 Ornaments

2.3.3 Functorial Species

2.3.4 Indexed Functors

2.4 Blockchain Semantics

2.4.1 BitML

2.4.2 UTXO & Extended UTXO

- Libraries for property based testing (QuickCheck, (Lazy) SmallCheck, QuickChick, QuickSpec)
- Type universes (ADT's, Ornaments) [5, 8]
- Generic programming techniques. (pattern functors, indexed functors, functorial species)
- Techniques to generate complex or constrained data (Generating constrained random data with uniform distribution, Generators for inductive relations)
- Techniques to speed up generation of data (Memoization, FEAT)

- Formal specification of blockchain (bitml, (extended) UTxO ledger) [15, 16]
- Representing potentially infinite data in Agda (Colists, coinduction, sized types)

Below is a bit of Agda code:

```
Γ-match : (τ : Ty) → ⟨⟨ ωᵢ (λ Γ → Σ[ α ∈ Id ] Γ [ α ↦ τ ]) ⟩⟩
Γ-match τ μ ∅ = uninhabited
Γ-match τ μ (α ↦ σ :: Γ) with τ ≟ σ
Γ-match τ μ (α ↦ τ :: Γ)  | yes refl  = ⦇ (α , TOP)              ⦈
                                     ∥ ⦇ (Σ-map POP) (μ Γ) ⦈
Γ-match τ μ (α ↦ σ :: Γ)  | no ¬p     = ⦇ (Σ-map POP) (μ Γ) ⦈
```

Listing 1: Definition of Γ-match

```
data Env : Set where
  ∅ : Env
  _↦_::_ : Id → Ty → Env → Env


data _[_↦_] : Env → Id → Ty → Set where

  TOP : ∀ {Γ α τ}
        → (α ↦ τ :: Γ) [ α ↦ τ ]

  POP : ∀ {Γ α β τ σ} → Γ [ α ↦ τ ]
        → (β ↦ σ :: Γ) [ α ↦ τ ]
```

Listing 2: Envirionment definition and membership in *Agda*

3 Preliminary results

What examples can you handle already? [9]

What prototype have I built? [4, 7]

How can I generalize these results? What problems have I identified or do I expect? [13]

$$TOP\ \frac{}{(a \mapsto t : \Gamma)[a \mapsto t]} \qquad POP\ \frac{\Gamma[a \mapsto t]}{(b \mapsto s : \Gamma)[a \mapsto t]}$$

$$VAR\ \frac{\Gamma[a \mapsto \tau]}{\Gamma \vdash a : \tau} \qquad ABS\ \frac{\Gamma, a \mapsto \sigma \vdash t : \tau}{\Gamma \vdash \lambda a \to t : \sigma \to \tau}$$

$$APP\ \frac{\Gamma \vdash f : \sigma \to \tau \quad \Gamma \vdash x : \sigma}{\Gamma \vdash fx : \tau} \qquad LET\ \frac{\Gamma \vdash e : \sigma \quad \Gamma, a \mapsto \sigma \vdash t : \tau}{\Gamma \vdash \ \texttt{let}\ a := e\ \texttt{in}\ t : \tau}$$

Listing 3: Semantics of the *Simply Typed Lambda Calculus*

4 Timetable and planning

What will I do with the remainder of my thesis? [2]

Give an approximate estimation/timetable for what you will do and when you will be done.

References

[1] Thorsten Altenkirch and Conor McBride. Generic programming within dependently typed programming. In *Generic Programming*, pages 1–20. Springer, 2003.

[2] Koen Claessen, Jonas Duregård, and Michał H Pałka. Generating constrained random data with uniform distribution. *Journal of functional programming*, 25, 2015.

[3] Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of haskell programs. *Acm sigplan notices*, 46(4):53–64, 2011.

[4] Koen Claessen, Nicholas Smallbone, and John Hughes. Quickspec: Guessing formal specifications using testing. In *International Conference on Tests and Proofs*, pages 6–21. Springer, 2010.

[5] Pierre-Évariste Dagand. The essence of ornaments. *Journal of Functional Programming*, 27, 2017.

[6] Maxime Dénès, Catalin Hritcu, Leonidas Lampropoulos, Zoe Paraskevopoulou, and Benjamin C Pierce. Quickchick: Property-based testing for coq. In *The Coq Workshop*, 2014.

[7] Jonas Duregård, Patrik Jansson, and Meng Wang. Feat: functional enumeration of algebraic types. *ACM SIGPLAN Notices*, 47(12):61–72, 2013.

[8] Hsiang-Shang Ko and Jeremy Gibbons. Programming with ornaments. *Journal of Functional Programming*, 27, 2016.

[9] Leonidas Lampropoulos, Zoe Paraskevopoulou, and Benjamin C Pierce. Generating good generators for inductive relations. *Proceedings of the ACM on Programming Languages*, 2(POPL):45, 2017.

[10] Andres Löh and José Pedro Magalhaes. Generic programming with indexed functors. In *Proceedings of the seventh ACM SIGPLAN workshop on Generic programming*, pages 1–12. ACM, 2011.

[11] Ulf Norell. Dependently typed programming in agda. In *International School on Advanced Functional Programming*, pages 230–266. Springer, 2008.

[12] Colin Runciman, Matthew Naylor, and Fredrik Lindblad. Smallcheck and lazy smallcheck: automatic exhaustive testing for small values. In *Acm sigplan notices*, volume 44, pages 37–48. ACM, 2008.

[13] Alexey Rodriguez Yakushev, Stefan Holdermans, Andres Löh, and Johan Jeuring. Generic programming with fixed points for mutually recursive datatypes. In *ACM Sigplan Notices*, volume 44, pages 233–244. ACM, 2009.

[14] Brent A Yorgey. Species and functors and types, oh my! In *ACM Sigplan Notices*, volume 45, pages 147–158. ACM, 2010.

[15] Joachim Zahnentferner. An abstract model of utxo-based cryptocurrencies with scripts. *IACR Cryptology ePrint Archive*, 2018:469, 2018.

[16] Joachim Zahnentferner and Input Output HK. Chimeric ledgers: Translating and unifying utxo-based and account-based cryptocurrencies. Technical report, Cryptology ePrint Archive, Report 2018/262, 2018. https://epri nt. iacr. org . . . , 2018.