Implementation of static and dynamic semantics for a calculus with algebraic effects and handlers using PLT Redex

Maciej Buszka

Instytut Informatyki UWr

15.02.2019

Plan prezentacji

Efekty algebraiczne

Efekty algebraiczne

- Na wysokim poziomie pozwalają na:

Efekty algebraiczne

- Na wysokim poziomie pozwalają na:
  - Kompozycję wielu różnych efektów obliczeniowych

Efekty algebraiczne

- Na wysokim poziomie pozwalają na:
  - Kompozycję wielu różnych efektów obliczeniowych
  - Definicję własnych efektów przez programistę

Efekty algebraiczne

- Na wysokim poziomie pozwalają na:
  - Kompozycję wielu różnych efektów obliczeniowych
  - Definicję własnych efektów przez programistę
  - Rozdzielenie interfejsu i implementacji

Efekty algebraiczne

- Na wysokim poziomie pozwalają na:
  - Kompozycję wielu różnych efektów obliczeniowych
  - Definicję własnych efektów przez programistę
  - Rozdzielenie interfejsu i implementacji
- W użyciu:

Efekty algebraiczne

- Na wysokim poziomie pozwalają na:
  - Kompozycję wielu różnych efektów obliczeniowych
  - Definicję własnych efektów przez programistę
  - Rozdzielenie interfejsu i implementacji
- W użyciu:
  - Pisząc program/funkcję programista może użyć abstrakcyjnych operacji – ich zbiór tworzy interfejs

Efekty algebraiczne

- Na wysokim poziomie pozwalają na:
  - Kompozycję wielu różnych efektów obliczeniowych
  - Definicję własnych efektów przez programistę
  - Rozdzielenie interfejsu i implementacji
- W użyciu:
  - Pisząc program/funkcję programista może użyć abstrakcyjnych operacji – ich zbiór tworzy interfejs
  - Aby uruchomić taki program, musi on zostać umieszczony w wyrażeniu które te operacje obsłuży – tworząc ich implementację

Obsługa operacji

Obsługa operacji

W trakcie wykonania programu, gdy operacja zostaje wywołana:

Obsługa operacji

W trakcie wykonania programu, gdy operacja zostaje wywołana:

- Normalny tok obliczeń zostaje przerwany

Obsługa operacji

W trakcie wykonania programu, gdy operacja zostaje wywołana:

- Normalny tok obliczeń zostaje przerwany
- Odnalezione zostaje odpowiednie wyrażenie obsługujące

Obsługa operacji

W trakcie wykonania programu, gdy operacja zostaje wywołana:

- Normalny tok obliczeń zostaje przerwany
- Odnalezione zostaje odpowiednie wyrażenie obsługujące
- Ma ono dostęp do:

Obsługa operacji

W trakcie wykonania programu, gdy operacja zostaje wywołana:

- Normalny tok obliczeń zostaje przerwany
- Odnalezione zostaje odpowiednie wyrażenie obsługujące
- Ma ono dostęp do:
  - Wartości przekazanej przy wywołaniu operacji

Obsługa operacji

W trakcie wykonania programu, gdy operacja zostaje wywołana:

- Normalny tok obliczeń zostaje przerwany
- Odnalezione zostaje odpowiednie wyrażenie obsługujące
- Ma ono dostęp do:
  - Wartości przekazanej przy wywołaniu operacji
  - Funkcji reprezentującej resztę obliczenia które zostało przerwane

Obsługa operacji

W trakcie wykonania programu, gdy operacja zostaje wywołana:

- Normalny tok obliczeń zostaje przerwany
- Odnalezione zostaje odpowiednie wyrażenie obsługujące
- Ma ono dostęp do:
  - Wartości przekazanej przy wywołaniu operacji
  - Funkcji reprezentującej resztę obliczenia które zostało przerwane
- Wyrażenie obsługujące może użyć wznowienia dowolnie wiele razy

Przykład 1

Przykład 1

```
(handle
  (+ (op:Magic (- 2 1)) 1)
  ((op:Magic (v:x v:r (app v:r 42))))
  (return v:x -(v:x, 1))
```

Przykład 1

```
(handle
  (+ (op:Magic (- 2 1)) 1)
  ((op:Magic (v:x v:r (app v:r 42))))
  (return v:x -(v:x, 1))
```

Przykład 1

```
(handle
  (+ (op:Magic 1) 1)
  ((op:Magic (v:x v:r (app v:r 42))))
  (return v:x -(v:x, 1))
```

Przykład 1

```
(handle
  (+ (op:Magic 1) 1)
  ((op:Magic (v:x v:r (app v:r 42))))
  (return v:x -(v:x, 1))
```

Przykład 1

```
(app
  (λ v:z
    (handle
      (+ v:z 1)
      (...)
      (return v:x -(v:x, 1))))
  42)
```

Przykład 1

```
(app
  (λ v:z
    (handle
      (+ v:z 1)
      (...)
      (return v:x -(v:x, 1))))
  42)
```

Przykład 1

```
(handle
  (+ 42 1)
  ( ... )
  (return v:x«38» -(v:x«38», 1)))
```

Przykład 1

```
(handle
  (+ 42 1)
  ( ... )
  (return v:x«38» -(v:x«38», 1)))
```

Przykład 1

```
(handle
  43
  ( ... )
  (return v:x«38» -(v:x«38», 1)))
```

Przykład 1

```
(handle
  43
  ( ... )
  (return v:x«38» -(v:x«38», 1)))
```

Przykład 1

```
-(43, 1)
```

Przykład 1

```
-(43, 1)
```

Przykład 1

42

Motywacja

Motywacja

- Efekty algebraiczne są stosunkowo nowym zagadnieniem

Motywacja

- Efekty algebraiczne są stosunkowo nowym zagadnieniem
- Ich implementacja oraz semantyka (zarówno statyczna jak i dynamiczna) są nietrywialne

Motywacja

- Efekty algebraiczne są stosunkowo nowym zagadnieniem
- Ich implementacja oraz semantyka (zarówno statyczna jak i dynamiczna) są nietrywialne
- Istnieje już kilka języków programowania z efektami algebraicznymi

Motywacja

- Efekty algebraiczne są stosunkowo nowym zagadnieniem
- Ich implementacja oraz semantyka (zarówno statyczna jak i dynamiczna) są nietrywialne
- Istnieje już kilka języków programowania z efektami algebraicznymi
- Brakuje jednak interaktywnej implementacji modelowego rachunku

Cel

Moim celem było zaprojektowanie oraz implementacja:

Cel

Moim celem było zaprojektowanie oraz implementacja:

- Rachunku oraz jego semantyki dynamicznej

Cel

Moim celem było zaprojektowanie oraz implementacja:

- Rachunku oraz jego semantyki dynamicznej
- Semantyki statycznej – systemu typów

Cel

Moim celem było zaprojektowanie oraz implementacja:

- Rachunku oraz jego semantyki dynamicznej
- Semantyki statycznej – systemu typów
- Maszyny abstrakcyjnej zgodnej z semantyka dynamiczną

Rachunek

Rachunek

Rachunek który zaimplementowałem udostępnia:

Rachunek

Rachunek który zaimplementowałem udostępnia:

- Wyrażenia liczbowe z podstawowymi operacjami

Rachunek

Rachunek który zaimplementowałem udostępnia:

- Wyrażenia liczbowe z podstawowymi operacjami
- Wyrażenia logiczne i warunkowe

Rachunek

Rachunek który zaimplementowałem udostępnia:

- Wyrażenia liczbowe z podstawowymi operacjami
- Wyrażenia logiczne i warunkowe
- Homogeniczne listy (dla dowolnego typu)

Rachunek

Rachunek który zaimplementowałem udostępnia:

- Wyrażenia liczbowe z podstawowymi operacjami
- Wyrażenia logiczne i warunkowe
- Homogeniczne listy (dla dowolnego typu)
- Funkcje anonimowe oraz rekurencyjne

Rachunek

Efekty algebraiczne są realizowane przez:

Rachunek

Efekty algebraiczne są realizowane przez:

- Abstrakcyjne operacje

Rachunek

Efekty algebraiczne są realizowane przez:

- Abstrakcyjne operacje
  - Nie muszą być zdefiniowane a priori

Rachunek

Efekty algebraiczne są realizowane przez:

- Abstrakcyjne operacje
  - Nie muszą być zdefiniowane a priori
  - Przyjmują jeden argument, zwracają (być może) jedną wartość

Rachunek

Efekty algebraiczne są realizowane przez:

- Abstrakcyjne operacje
  - Nie muszą być zdefiniowane a priori
  - Przyjmują jeden argument, zwracają (być może) jedną wartość
- Wyrażenia obsługujące operacje

Rachunek

Efekty algebraiczne są realizowane przez:

- Abstrakcyjne operacje
  - Nie muszą być zdefiniowane a priori
  - Przyjmują jeden argument, zwracają (być może) jedną wartość
- Wyrażenia obsługujące operacje
  - Dostęp do wznowienia reifikowanego jako funkcja

Rachunek

Efekty algebraiczne są realizowane przez:

- Abstrakcyjne operacje
  - Nie muszą być zdefiniowane a priori
  - Przyjmują jeden argument, zwracają (być może) jedną wartość
- Wyrażenia obsługujące operacje
  - Dostęp do wznowienia reifikowanego jako funkcja
  - Semantyka tzw. głębokiej obsługi

Rachunek

Efekty algebraiczne są realizowane przez:

- Abstrakcyjne operacje
  - Nie muszą być zdefiniowane a priori
  - Przyjmują jeden argument, zwracają (być może) jedną wartość
- Wyrażenia obsługujące operacje
  - Dostęp do wznowienia reifikowanego jako funkcja
  - Semantyka tzw. głębokiej obsługi
  - Obsługa wielu operacji naraz

System typów

System typów

- Implementacja tworzy funkcję odtwarzającą typ wyrażenia

System typów

- Implementacja tworzy funkcję odtwarzającą typ wyrażenia
- Inferowane są tylko typy proste oraz operacje

System typów

- Implementacja tworzy funkcję odtwarzającą typ wyrażenia
- Inferowane są tylko typy proste oraz operacje
- Nie ma możliwości stworzenia funkcji polimorficznych

System typów

- Implementacja tworzy funkcję odtwarzającą typ wyrażenia
- Inferowane są tylko typy proste oraz operacje
- Nie ma możliwości stworzenia funkcji polimorficznych
- Ale system znajduje najogólniejszy (prosty) typ wyrażenia

Przykład 2

Przykład 2

Wyrażenie:

Przykład 2

Wyrażenie:

```
λ ignore
  let a = Set 13 in
  +(Get 0, Get (Set 29))
```

Przykład 2

Wyrażenie:

```
λ ignore
  let a = Set 13 in
  +(Get 0, Get (Set 29))
```

Ma typ:

Przykład 2

Wyrażenie:

```
λ ignore
  let a = Set 13 in
  +(Get 0, Get (Set 29))
```

Ma typ:

```
(t:g0 -> (op:Set (Num => Num)
          (op:Get (Num => Num) t:r14)) Num)
```