

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

Maciej Buszka

Instytut Informatyki UW

15.02.2019

Plan prezentacji

- 1 Wstęp
 - Cel
 - Efekty algebraiczne
- 2 Rachunek oraz model
 - Rachunek
 - Model
- 3 Podsumowanie

Motywacja i cel pracy

- Nowe, aktywnie rozwijane zagadnienie

Motywacja i cel pracy

- Nowe, aktywnie rozwijane zagadnienie
- Nietrywialna semantyka i implementacja

Motywacja i cel pracy

- Nowe, aktywnie rozwijane zagadnienie
- Nietrywialna semantyka i implementacja
- Brak interaktywnego modelu rachunku

Motywacja i cel pracy

- Nowe, aktywnie rozwijane zagadnienie
- Nietrywialna semantyka i implementacja
- Brak interaktywnego modelu rachunku
- Cele:

Motywacja i cel pracy

- Nowe, aktywnie rozwijane zagadnienie
- Nietrywialna semantyka i implementacja
- Brak interaktywnego modelu rachunku
- Cele:
 - Zaprojektowanie rachunku oraz jego semantyki

Motywacja i cel pracy

- Nowe, aktywnie rozwijane zagadnienie
- Nietrywialna semantyka i implementacja
- Brak interaktywnego modelu rachunku
- Cele:
 - Zaprojektowanie rachunku oraz jego semantyki
 - Implementacja modelu

Efekty algebraiczne

- Kompozycja wielu różnych efektów obliczeniowych

Efekty algebraiczne

- Kompozycja wielu różnych efektów obliczeniowych
- Definicja własnych efektów

Efekty algebraiczne

- Kompozycja wielu różnych efektów obliczeniowych
- Definicja własnych efektów
- Rozdzielenie interfejsu i implementacji

Efekty algebraiczne

- Kompozycja wielu różnych efektów obliczeniowych
- Definicja własnych efektów
- Rozdzielenie interfejsu i implementacji
- Program może używać abstrakcyjnych operacji – interfejs

Efekty algebraiczne

- Kompozycja wielu różnych efektów obliczeniowych
- Definicja własnych efektów
- Rozdzielenie interfejsu i implementacji
- Program może używać abstrakcyjnych operacji – interfejs
- Wyrażenie obsługujące – implementacja

Rachunek

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

Rachunek

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

Rachunek

- Abstrakcyjne operacje
 - Nie muszą być zdefiniowane a priori
 - Przyjmują jeden argument, zwracają jedną wartość
- Wyrażenia obsługujące (*handler*)
 - Dostęp do wznowienia
 - Semantyka tzw. głębokiej obsługi
 - Obsługa wielu operacji naraz
 - Klauzula return
- Wyrażenia podnoszące (*lift*)
 - 'przeskoczenie' najbliższego wyrażenia obsługującego

Rachunek

- Abstrakcyjne operacje
 - Nie muszą być zdefiniowane a priori
 - Przyjmują jeden argument, zwracają jedną wartość
- Wyrażenia obsługujące (*handler*)
 - Dostęp do wznowienia
 - Semantyka tzw. głębokiej obsługi
 - Obsługa wielu operacji naraz
 - Klauzula return
- Wyrażenia podnoszące (*lift*)
 - 'przeskoczenie' najbliższego wyrażenia obsługującego
- Wartości bazowe oraz operacje na nich

Model

- Składnia abstrakcyjna
- Semantyka redukcyjna

Model

- Składnia abstrakcyjna
- Semantyka redukcyjna
- System typów
 - Odtwarza typ i efekt wyrażenia
 - Polimorfizm nie jest wspierany
 - Najogólniejszy typ wyrażenia

Model

- Składnia abstrakcyjna
- Semantyka redukcyjna
- System typów
 - Odtwarza typ i efekt wyrażenia
 - Polimorfizm nie jest wspierany
 - Najogólniejszy typ wyrażenia
- Maszyna abstrakcyjna
 - Jawne środowisko
 - Stos i meta-stos

Działanie modelu

- Wizualizacja redukcji
- Przykładowy program

```
1  #lang algeff
2
3  (handle +(Get 0, Get (Set 29)) with
4    | Get i r -> λ s (r s s)
5    | Set s r -> λ i (r 0 s)
6    | return x -> λ s x
7  end) 13
```

- Oczekiwany wynik: 42

Działanie modelu

```
1  (app
2    (handle
3      (+ (op:Get 0) (op:Get (op:Set 29)))
4      ((op:Get (v:i v:r (λ v:s (app (app v:r v:s) v:s)))))
5      (op:Set (v:s v:r (λ v:i (app (app v:r 0) v:s)))))
6      (return v:x (λ v:s v:x)))
7    13)
```

- Składnia abstrakcyjna:

- linia 1. app – aplikacja
- linia 3. wyrażenie zagnieżdżone
- linie 4. i 5. klauzule obsługujące

Działanie modelu

Wywołanie operacji Get

```
1  (app
2    (handle
3      (+ (op:Get 0) (op:Get (op:Set 29)))
4      ((op:Get (v:i v:r (λ v:s (app (app v:r v:s) v:s))))
5        (op:Set (v:s v:r (λ v:i (app (app v:r 0) v:s))))))
6      (return v:x (λ v:s v:x)))
7  13)
```

- kolor **pomarańczowy** – operacja i jej handler
- kolor **czerwony** – przechwycony kontekst

Działanie modelu

```
1  (app
2    (λ v:s«312»
3      (app
4        (app
5          (λ v:z
6            (handle
7              (+ v:z (op:Get (op:Set 29)))
8              ( ... )
9              (return v:x (λ v:s v:x))))
10         v:s«312»)
11         v:s«312»))
12  13)
```

- kolor **czzerwony** – przechwycony kontekst
- kolor **niebieski** – wyrażenie stworzone przez redukcję

Działanie modelu

β -redukcja

```
1  (app
2    (λ v:s«312»
3      (app (app
4        (λ v:z
5          (handle
6            (+ v:z (op:Get (op:Set 29)))
7            ( ... )
8            (return v:x (λ v:s v:x))))
9        v:s«312»)
10       v:s«312»))
11  13)
```

- kolor **pomarańczowy** – redeks

Działanie modelu

```
1  (app
2    (app
3      (λ v:z«315»
4        (handle
5          (+ v:z«315» (op:Get (op:Set 29)))
6          ( ... )
7          (return v:x«318» (λ v:s«319» v:x«318»))))
8    13)
9  13)
```

- kolor **niebieski** – wartości podstawione za zmienną

Działanie modelu

β -redukcja

```
1  (app
2    (app
3      ( $\lambda v:z\ll 315\gg$ 
4        (handle
5          (+  $v:z\ll 315\gg$  (op:Get (op:Set 29)))
6          ( ... )
7          (return  $v:x\ll 318\gg$  ( $\lambda v:s\ll 319\gg v:x\ll 318\gg$ ))))
8    13)
9  13)
```

- kolor **pomarańczowy** – redeks

Działanie modelu

```
1  (app
2    (handle
3      (+ 13 (op:Get (op:Set 29)))
4      ( ... )
5      (return v:x«318» (λ v:s«319» v:x«318»))))
6  13)
```

- kolor **niebieski** – wartość podstawiona za zmienną

Działanie modelu

Wywołanie operacji Set

```
1 (app
2   (handle
3     (+ 13 (op:Get (op:Set 29)))
4     ((op:Get ( ... ))
5       (op:Set (v:s v:r (λ v:i«331» (app (app v:r 0) v:s))))))
6     (return v:x«332» (λ v:s«333» v:x«332»)))
7   13)
```

- kolor **pomarańczowy** – operacja i jej handler
- kolor **czerwony** – przechwycony kontekst

Działanie modelu

```
1  (app
2    (λ v:i«343»
3      (app
4        (app
5          (λ v:z
6            (handle
7              (+ 13 (op:Get v:z))
8              ( ... )
9              (return v:x«332» (λ v:s«333» v:x«332»))))
10         0)
11     29))
12  13)
```

- kolor **czzerwony** – przechwycony kontekst
- kolor **niebieski** – wyrażenie stworzone przez redukcję

Działanie modelu

β -redukcja

```
1  (app
2    (λ v:i«343»
3      (app
4        (app
5          (λ v:z
6            (handle
7              (+ 13 (op:Get v:z))
8              ( ... )
9              (return v:x«332» (λ v:s«333» v:x«332»))))
10         0)
11       29))
12  13)
```

- kolor **pomarańczowy** – redeks

Działanie modelu

```
1  (app
2    (app
3      (λ v:z
4        (handle
5          (+ 13 (op:Get v:z))
6          ( ... )
7          (return v:x«332» (λ v:s«333» v:x«332»))))))
8    0)
9  29)
```


Działanie modelu

β -redukcja

```
1  (app
2    (app
3      ( $\lambda$  v:z
4        (handle
5          (+ 13 (op:Get v:z))
6          ( ... )
7          (return v:x«332» ( $\lambda$  v:s«333» v:x«332»))))
8    0)
9  29)
```

- kolor pomarańczowy – redeks

Działanie modelu

```
1  (app
2    (handle
3      (+ 13 (op:Get 0))
4      ( ... )
5      (return v:x«332» (λ v:s«333» v:x«332»)))
6    29)
```

- kolor **niebieski** – wartość podstawiona za zmienną

Działanie modelu

Wywołanie operacji Get

```
1  (app
2    (handle
3      (+ 13 (op:Get 0))
4      ((op:Get (v:i v:r (λ v:s (app (app v:r v:s) v:s))))
5        (op:Set ( ... ))
6        (return v:x«332» (λ v:s«333» v:x«332»))))
7  29)
```

- kolor **pomarańczowy** – operacja i jej handler
- kolor **czerwony** – przechwycony kontekst

Działanie modelu

```
1  (app
2    (handle
3      (+ 13 29)
4      ( ... )
5      (return v:x«332» (λ v:s«333» v:x«332»)))
6    29)
```

- kolor **niebieski** – wynik po trzech krokach redukcji

Działanie modelu

Operacja bazowa

```
1 (app
2   (handle
3     (+ 13 29)
4     ( ... )
5     (return v:x«332» (λ v:s«333» v:x«332»)))
6   29)
```

- kolor **pomarańczowy** – operacja bazowa

Działanie modelu

```
1  (app
2    (handle
3      42
4      ( ... )
5      (return v:x«332» (λ v:s«333» v:x«332»)))
6    29)
```

- kolor **niebieski** – wynik operacji

Działanie modelu

Obsługa wartości

```
1 (app
2   (handle
3     42
4     ( ... )
5     (return v:x«332» (λ v:s«333» v:x«332»)))
6   29)
```

- kolor **pomarańczowy** – wartość i klauzula return

Działanie modelu

```
1  (app
2    (λ v:s«405» 42)
3    29)
```

- kolor **niebieski** – wyrażenie stworzone przez redukcję

Działanie modelu

β -redukcja

```
1  (app
2    ( $\lambda v:s\langle 405 \rangle$  42)
3    29)
```

- kolor **pomarańczowy** – redeks

Działanie modelu

- Ostateczny wynik: 42
- Redukcja w 13 krokach

Działanie modelu

- Inferencja typów

Działanie modelu

- Inferencja typów
- Program:

```
#lang algeff
```

```
λ ignore  
  +(Get 0, Get (Set 29))
```

Działanie modelu

- Inferencja typów
- Program:

```
#lang algeff
```

```
λ ignore  
  +(Get 0, Get (Set 29))
```

- Typ:

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

Podsumowanie

- Rachunek z efektami algebraicznymi
 - Operacje, wyrażenia obsługujące i podnoszące
 - Wyrażenia ogólnego zastosowania

Podsumowanie

- Rachunek z efektami algebraicznymi
 - Operacje, wyrażenia obsługujące i podnoszące
 - Wyrażenia ogólnego zastosowania
- Implementacja modelu
 - Inferencja typu i efektu
 - Pełna redukcja wyrażeń
 - Wizualizacja krok po kroku

Podsumowanie

- Rachunek z efektami algebraicznymi
 - Operacje, wyrażenia obsługujące i podnoszące
 - Wyrażenia ogólnego zastosowania
- Implementacja modelu
 - Inferencja typu i efektu
 - Pełna redukcja wyrażeń
 - Wizualizacja krok po kroku
 - Front-end – *algeff*
 - Integracja ze środowiskiem Racket

Podsumowanie

- Rachunek z efektami algebraicznymi
 - Operacje, wyrażenia obsługujące i podnoszące
 - Wyrażenia ogólnego zastosowania
- Implementacja modelu
 - Inferencja typu i efektu
 - Pełna redukcja wyrażeń
 - Wizualizacja krok po kroku
 - Front-end – *algeff*
 - Integracja ze środowiskiem Racket
- Rozszerzenie rachunku o polimorfizm

Podsumowanie

- Rachunek z efektami algebraicznymi
 - Operacje, wyrażenia obsługujące i podnoszące
 - Wyrażenia ogólnego zastosowania
- Implementacja modelu
 - Inferencja typu i efektu
 - Pełna redukcja wyrażeń
 - Wizualizacja krok po kroku
 - Front-end – *algeff*
 - Integracja ze środowiskiem Racket
- Rozszerzenie rachunku o polimorfizm
- Zbadanie własności rachunku za pomocą automatycznego generowania programów

Dziękuję za uwagę

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