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

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
- Cel – zaprojektowanie i implementacja:

Motywacja i cel pracy

- Nowe, aktywnie rozwijane zagadnienie
- Nietrywialna semantyka i implementacja
- Brak interaktywnego modelu rachunku
- Cel – zaprojektowanie i implementacja:
  - Rachunku oraz jego semantyki dynamicznej

Motywacja i cel pracy

- Nowe, aktywnie rozwijane zagadnienie
- Nietrywialna semantyka i implementacja
- Brak interaktywnego modelu rachunku
- Cel – zaprojektowanie i implementacja:
  - Rachunku oraz jego semantyki dynamicznej
  - Semantyki statycznej

Motywacja i cel pracy

- Nowe, aktywnie rozwijane zagadnienie
- Nietrywialna semantyka i implementacja
- Brak interaktywnego modelu rachunku
- Cel – zaprojektowanie i implementacja:
  - Rachunku oraz jego semantyki dynamicznej
  - Semantyki statycznej
  - Maszyny abstrakcyjnej

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

Rachunek

- Abstrakcyjne operacje
  - Nie muszą być zdefiniowane a priori

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

Rachunek

- Abstrakcyjne operacje
  - Nie muszą być zdefiniowane a priori
  - Przyjmują jeden argument, zwracają jedną wartość
- Wyrażenia obsługujące
  - Dostęp do wznowienia

Rachunek

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

Rachunek

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

Rachunek

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

Rachunek

- Abstrakcyjne operacje
  - Nie muszą być zdefiniowane a priori
  - Przyjmują jeden argument, zwracają jedną wartość
- Wyrażenia obsługujące
  - 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
  - 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 i semantyka dynamiczna

Model

- Składnia abstrakcyjna i semantyka dynamiczna
- System typów
  - Odtwarza typ i efekt wyrażenia

Model

- Składnia abstrakcyjna i semantyka dynamiczna
- System typów
  - Odtwarza typ i efekt wyrażenia
  - Inferowane są tylko typy proste

Model

- Składnia abstrakcyjna i semantyka dynamiczna
- System typów
  - Odtwarza typ i efekt wyrażenia
  - Inferowane są tylko typy proste
  - Polimorfizm nie jest wspierany

Model

- Składnia abstrakcyjna i semantyka dynamiczna
- System typów
  - Odtwarza typ i efekt wyrażenia
  - Inferowane są tylko typy proste
  - Polimorfizm nie jest wspierany
  - Najogólniejszy typ wyrażenia

Model

- Składnia abstrakcyjna i semantyka dynamiczna
- System typów
  - Odtwarza typ i efekt wyrażenia
  - Inferowane są tylko typy proste
  - Polimorfizm nie jest wspierany
  - Najogólniejszy typ wyrażenia
- Maszyna abstrakcyjna

Model

- Składnia abstrakcyjna i semantyka dynamiczna
- System typów
  - Odtwarza typ i efekt wyrażenia
  - Inferowane są tylko typy proste
  - Polimorfizm nie jest wspierany
  - Najogólniejszy typ wyrażenia
- Maszyna abstrakcyjna
  - Wiele konfiguracji

Model

- Składnia abstrakcyjna i semantyka dynamiczna
- System typów
  - Odtwarza typ i efekt wyrażenia
  - Inferowane są tylko typy proste
  - Polimorfizm nie jest wspierany
  - Najogólniejszy typ wyrażenia
- Maszyna abstrakcyjna
  - Wiele konfiguracji
  - Jawne środowisko

Model

- Składnia abstrakcyjna i semantyka dynamiczna
- System typów
  - Odtwarza typ i efekt wyrażenia
  - Inferowane są tylko typy proste
  - Polimorfizm nie jest wspierany
  - Najogólniejszy typ wyrażenia
- Maszyna abstrakcyjna
  - Wiele konfiguracji
  - Jawne środowisko
  - Stos i meta-stos

Przykład

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
```

Przykład

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

Przykład

```
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:

Przykład

```
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

Przykład

```
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

Przykład

```
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

Przykład

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)
```

Przykład

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

Przykład

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

Przykład

```
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 czerwony – przechwycony kontekst
- kolor niebieski – wyrażenie stworzone przez redukcję

Przykład

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

Przykład

```
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

Przykład

β-redukcja

```
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 pomarańczowy – redeks

Przykład

```
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

Przykład

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

Przykład

```
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 czerwony – przechwycony kontekst
- kolor niebieski – wyrażenie stworzone przez redukcję

Przykład

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

Przykład

```
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)
```

Przykład

β-redukcja

```
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)
```

- kolor pomarańczowy – redeks

Przykład

```
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

Przykład

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

Przykład

```
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

Przykład

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

Przykład

```
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

Przykład

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`

Przykład

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

- kolor niebieski – wyrażenie stworzone przez redukcję

Przykład

β-redukcja

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

- kolor pomarańczowy – redeks

Przykład

```
1    42
```

Przykład

Program:

```
#lang algeff

λ ignore
  +(Get 0, Get (Set 29))
```

Przykład

Program:

```
#lang algeff

λ ignore
  +(Get 0, Get (Set 29))
```

Ma typ:

Przykład

Program:

```
#lang algeff

λ ignore
  +(Get 0, Get (Set 29))
```

Ma typ:

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

Przykład

Program:

```
#lang algeff

handle +(Tick 0,
  handle +(Tick 0, lift Tock (Tock 1)) with
  | Tick x r -> r 0
  | Tock x r -> 12
  | return x -> x
  end) with
| Tick x r -> r 2
| Tock x r -> r 40
| return x -> x
end
```

Oczekiwany wynik: 42

Sukcesy

- Rachunek wraz z semantyką redukcyjną

Sukcesy

- Rachunek wraz z semantyką redukcyjną
  - Pełna redukcja wyrażeń

Sukcesy

- Rachunek wraz z semantyką redukcyjną
  - Pełna redukcja wyrażeń
  - Wizualizacja krok po kroku

Sukcesy

- Rachunek wraz z semantyką redukcyjną
  - Pełna redukcja wyrażeń
  - Wizualizacja krok po kroku
- Inferencja typów

Sukcesy

- Rachunek wraz z semantyką redukcyjną
  - Pełna redukcja wyrażeń
  - Wizualizacja krok po kroku
- Inferencja typów
- Maszyna abstrakcyjna

Sukcesy

- Rachunek wraz z semantyką redukcyjną
  - Pełna redukcja wyrażeń
  - Wizualizacja krok po kroku
- Inferencja typów
- Maszyna abstrakcyjna
  - Pełna redukcja wyrażeń

Sukcesy

- Rachunek wraz z semantyką redukcyjną
  - Pełna redukcja wyrażeń
  - Wizualizacja krok po kroku
- Inferencja typów
- Maszyna abstrakcyjna
  - Pełna redukcja wyrażeń
  - Wizualizacja krok po kroku

Sukcesy

- Rachunek wraz z semantyką redukcyjną
  - Pełna redukcja wyrażeń
  - Wizualizacja krok po kroku
- Inferencja typów
- Maszyna abstrakcyjna
  - Pełna redukcja wyrażeń
  - Wizualizacja krok po kroku
- Front-end ułatwiający pisanie wyrażeń

Sukcesy

- Rachunek wraz z semantyką redukcyjną
  - Pełna redukcja wyrażeń
  - Wizualizacja krok po kroku
- Inferencja typów
- Maszyna abstrakcyjna
  - Pełna redukcja wyrażeń
  - Wizualizacja krok po kroku
- Front-end ułatwiający pisanie wyrażeń
- Integracja ze środowiskiem `Racket`

Wnioski i dalsza praca

Wnioski:

- Wizualizacja jest bardzo pomocna przy badaniu efektów algebraicznych

Wnioski i dalsza praca

Wnioski:

- Wizualizacja jest bardzo pomocna przy badaniu efektów algebraicznych
- Biblioteka `PLT Redex` pozwala na rozwój rozbudowanego rachunku

Wnioski i dalsza praca

Wnioski:

- Wizualizacja jest bardzo pomocna przy badaniu efektów algebraicznych
- Biblioteka `PLT Redex` pozwala na rozwój rozbudowanego rachunku

Dalsza praca:

Wnioski i dalsza praca

Wnioski:

- Wizualizacja jest bardzo pomocna przy badaniu efektów algebraicznych
- Biblioteka `PLT Redex` pozwala na rozwój rozbudowanego rachunku

Dalsza praca:

- Rozszerzenie rachunku o polimorfizm

Wnioski i dalsza praca

Wnioski:

- Wizualizacja jest bardzo pomocna przy badaniu efektów algebraicznych
- Biblioteka `PLT Redex` pozwala na rozwój rozbudowanego rachunku

Dalsza praca:

- 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