Лаборатория языковых инструментов JetBRAINS

Объектно-ориентированные трансформеры для OCaml: анаморфизмы

Студент: Терешин Роман, 444 группа
Научный руководитель: к.ф.-м.н., доцент Д. Ю. Булычев
Рецензент: аспирант А. В. Подкопаев

Кафедра системного программирования, СПбГУ, 2015

Предметная область

- Алгебраические типы данных (ADT)
- Трансформеры ADT
- OCaml

Предметная область

- Катаморфизмы:

```
let rec fold step init = function
  | [] → init
  | x :: xs → fold step (step init x) xs

('b → 'a → 'b) → 'b → 'a list → 'b
```

- Анаморфизмы

```
let rec unfold step seed =
  match step seed with
  | Some (x, seed') -> x :: unfold step seed'
  | None -> []

('b → ('a * 'b) option) → 'b → 'a list
```

Возможные подходы

- Реализация “вручную”

- Фреймворк generic-transformers (Д. Ю. Булычев)
  - Управляемая типом генерация кода
  - Объектно-ориентированный
  - Набор типовых преобразователей
  - Встроен в язык
  - Только катаморфизмы

Постановка задачи

Цель:

- расширить OCaml поддержкой анаморфизмов

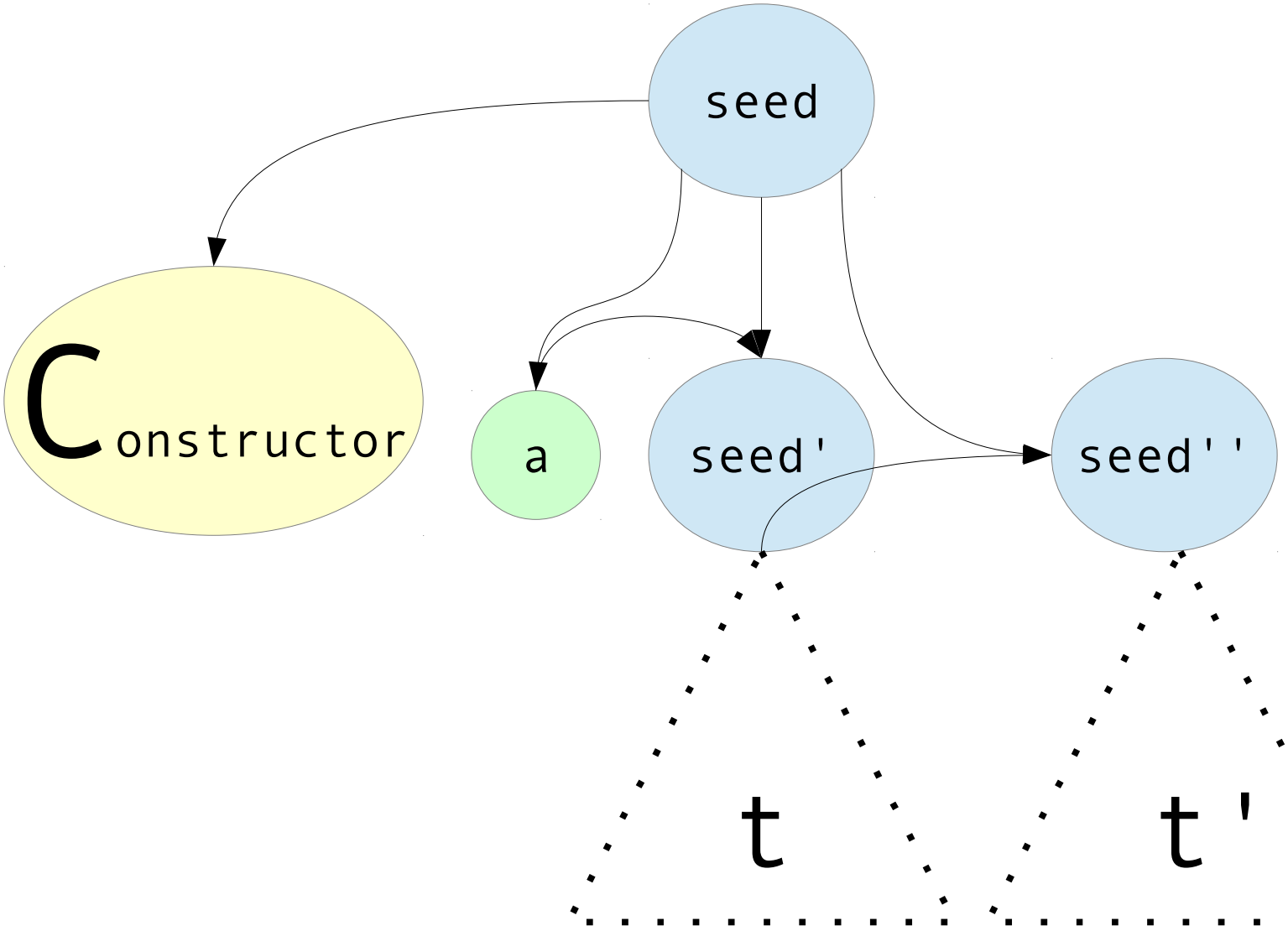
Задачи:

- Проанализировать существующие подходы
- Разработать семантику и интерфейс анаморфизмов
- Реализовать решение в рамках фреймворка
- Провести апробацию

Существующие подходы

- Директива deriving в Haskell
- Расширение deriving для OCaml (Yallop)
- Библиотека type_conv для OCaml (Jane Street)

Реализация (семантика)



интерфейс

- Пользовательский тип:

```
@type 'a t =
  | C_1 of p_11 * … * p_1n_1
  .       . . .
  | C_k of p_k1 * … * p_kn_k
```

- Тип состояния развертки:

```
type ('a, 'seed) state = [
  | `C_1 of q_11 * … * q_1n_1
  .       . . .
  | `C_k of q_k1 * … * q_1n_k
]
```

q_{ij} = ['seed →]$_{j>1}$ 'seed, если p_{ij} = t

q_{ij} = ['seed →]$_{j>1}$ p_{ij} * 'seed, иначе

интерфейс / генерация

- Тип пользовательской функции `('a, 'seed) step`:

```
('seed → 'a * 'seed) → 'seed → 'seed state
```

- Тип анаморфизма `unfold(t)`:

```
('seed → 'a * 'seed) →
  ('a, 'seed) step → 'seed → 'a t
```

- Генерация шаблонного кода: Camlp5

Апробация

```
open GT

@type 'a bst =
  | Null
  | Node of 'a * 'a bst * 'a bst

let bst_step = fun _ -> function
  | ([], _) -> `Null
  | (x :: xs, max) ->
      if x > max
      then `Null
      else `Node ((x, (xs, x)), pass, fun (tail, _) -> (tail, max))

let bst_of_preorder preorder =
  unfold'(bst) () bst_step (preorder, max_int)
```

Апробация

```
@type 'a lst =
  | Nil
  | Cons of 'a * 'a lst

@type 'expr_list expr =
  | Var of string
  | Constr of string * 'expr_list
  | Call of string * 'expr_list

@type ('id_list, 'expr_list) def =
  | FDef of string * 'id_list * 'expr_list expr
  | GDef of string * string * 'id_list * 'expr_list expr

type program = def' lst
```

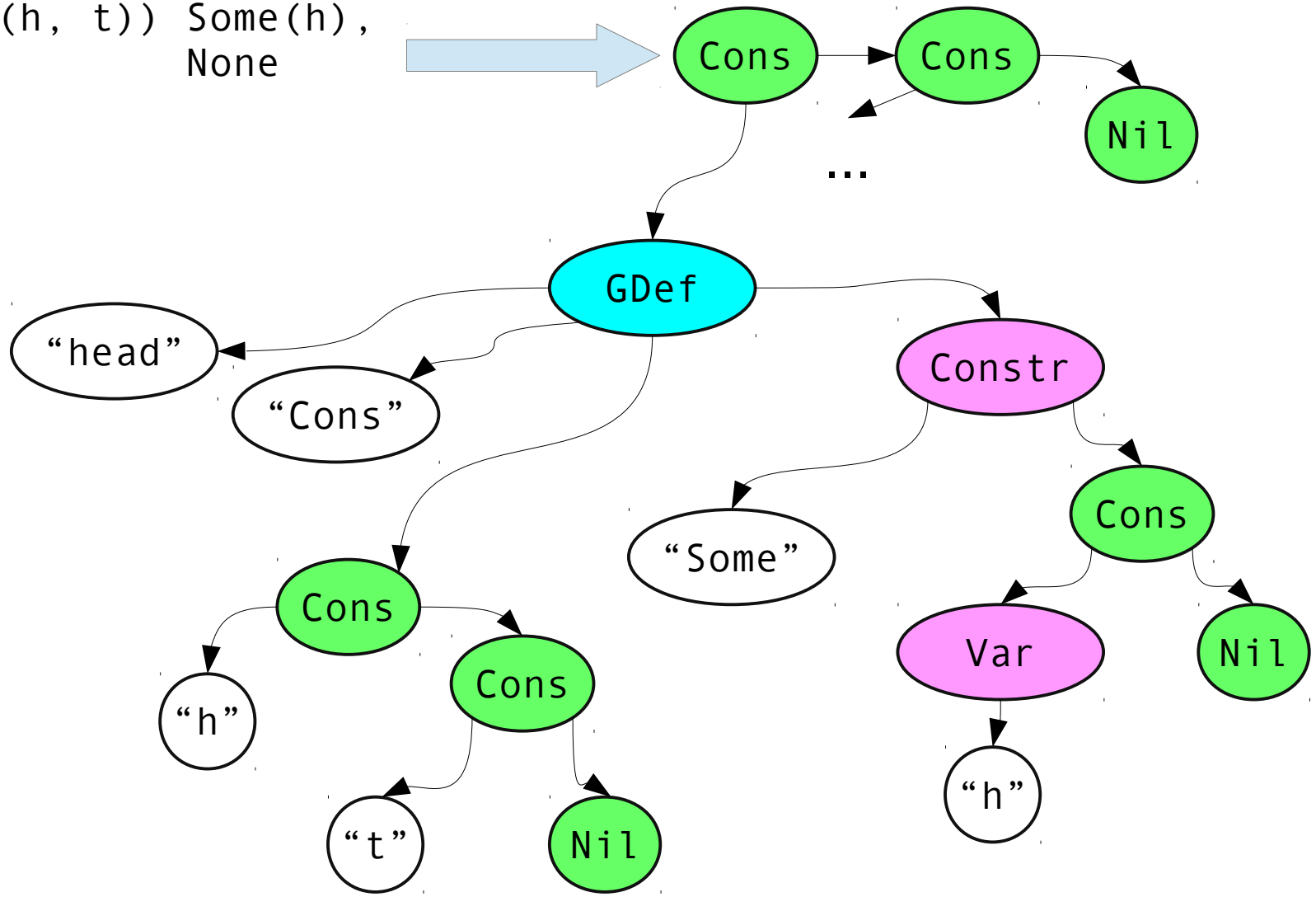
SLL

```
( head(Cons(h, t)) Some(h),
  head(Nil)        None
)
```

Размер реализации: в 1.5 раза меньше (LOC)

Апробация

```
( head(Cons(h, t)) Some(h),
  head(Nil)        None
)
```



Результаты

- Проанализированы существующие подходы
- Разработаны семантика и интерфейс анаморфизмов
- Реализована управляемая типом генерация шаблонного кода, реализация интегрирована с существующим фреймворком
- Проведена апробация