



Расширение сопоставления с образцом в языке OCaml

Автор: Александр Андреевич Башкиров, 17.Б11-мм

Научный руководитель: к.ф.-м.н, доцент, С.В. Григорьев

Консультант: программист “ИнтеллиДжей Лабс” Д.С. Косарев

Санкт-Петербургский государственный университет
Кафедра системного программирования

2 июня 2020 г.

Введение. Сопоставление с образцом

```
type 'a tree =  
  | Node of 'a tree * 'a * 'a tree  
  | Nil  
  
let rotate_left = function  
  | Node(a, p, Node(b, q, c)) -> Node(Node(a, p, b), q, c)  
  | Node _ | Nil as x -> x
```

Преимущества:

- высокая выразительность;
- **проверка на полноту**, являющаяся критически важным свойством для проверки корректности и рефакторинга программ;
- эффективные схемы компиляции

Возможно, поскольку известно определение типа дерева.

Сопоставление с образцом требует знания определения типа.

Введение. Постановка проблемы

```
module type List = sig
  type 'a t (* = ? *)
  val cons: 'a -> 'a t -> 'a t
  val empty: 'a t
  val head: 'a t -> ('a * 'a t) option
end
```

Структурами могут быть обычные списки, ленивые списки, списки на скошенных двоичных кучах, векторы, упорядоченные деревья и т.д.

Принципиально не можем экспортировать определение типа.

Следовательно не можем пользоваться сопоставлением с образцом.

Введение. Постановка проблемы

Недостатки:

- неприменимо для значений абстрактных типов;
- образцы не являются значениями первого класса языка

	Pattern matching	Абстракция	Инкапсуляция
Известный тип	+	—	—
Приватный тип	+	—	+
Абстрактный тип	—	+	+
Views 1987 [13]	+	+	+

Цель работы:

Разработать *консервативное* расширение языка OCaml, которое бы

- 1 позволяло использование сопоставление с образцом над значениями абстрактным типом;
- 2 являлось полностью статически типизируемым;
- 3 позволяло оперировать образцами как значениями первого порядка;
- 4 не требовало модификации существующих модулей

Задачи:

- 1 Провести анализ существующих решений и выполнить сравнение по ряду объективных критериев. По результатам обосновать выбор того или иного существующего решения или представить собственное.
- 2 Формализовать изменения, которые необходимо внести в компилятор языка OCaml для поддержки нового расширения:
 - ▶ расширение синтаксиса
 - ▶ эталонный типизирующий анализатор
 - ▶ компиляция в промежуточное представление
- 3 Представить прототип реализации выбранного решения.

Существующие решения

Год	Я. П.	Авторы	Статья
1987	Miranda	P. Wadler	Views [13]
1998	SML	C. Okasaki	Views for SML [8]
2000	Haskell	S.P. Jones et al.	Pattern guards [5]
2000	Haskell	M. Tullsen	First class patterns [12]
2000	Haskell	—	View patterns [14]
2007	Scala	M. Odersky et al.	Matching Objects with Patterns [4]
2007	F#	D. Syme et al.	Active Patterns [10]
2016	Haskell	S.P. Jones et al.	Pattern Synonyms [9]

Обзор существующих решений. Сравнение

Необходимо было представить *объективные критерии* сравнения дизайнов расширения конструкции сопоставления с образцом и произвести это сравнение.

Мотивация, разработка и обоснование представлены в [3]

Резюмируя:

- Одно из решений [12] было выбрано за эталонное по мощности.
- Введены формальные критерии **максимальности** и **выразительности** конструкции сопоставления с образцом.
- Произведен разбор существующих решений по данным критериям и разработано одно собственное (*оператор возврата к сопоставлению*).

Обзор существующих решений. Сравнение

Дизайн	Макс-ть	Выр-ть	1п.	Проверя-ть	Эф-ть
Views	—	—	—	+	?
Views for SML	—	—	—	+	+
Pattern guards	+	—	+	+	?
First class patterns	+	+	+	+/-	—
View patterns	+	—	+	+	?
Objects patterns	—	—	+	?	?
Active Patterns	+/-	—	+	+	+
Pattern Synonyms	—	—	?	+	?
Оператор возврата	+	+/-	+	+	+

Таблица 1: Сравнение конструкций сопоставления

* Во внимание не принималась интеграция с продвинутыми особенностями системы типов, таким как GADTs

Окончательный выбор дизайна

Явными фаворитами являются Оператор возврата и Активные образцы.

Совместно с сообществом было принято решение в пользу Активных образцов, как более известного и протестированного временем варианта.

По результатам был создан (пока не полноценный) RFC [1] по реализации активных образов в языке OCaml.

Наконец, сообщество ожидает прототип как образец синтаксиса и семантики, и базы для полноценной реализации.

Активные образцы

Активный образец представляет собой обычную функцию, отображающую значения абстрактного типа в некоторое известное представление, представленное в *структурированном имени*:

```
(* Proposed syntax for OCaml *)
```

```
let (|Cons|Nil|) l =  
  match head l with  
  | Some(hd, tl) -> Cons(hd, tl)  
  | None          -> Nil
```

```
let (|Apply|_|) f v = f v
```

```
let filter_map f = function  
  | Cons(<Apply f> r, xs) -> cons r (filter_map f xs)  
  | Cons(_, xs)          -> filter_map f xs  
  | Nil                  -> nil
```

Изменения синтаксиса

Для представления синтаксиса мы воспользуемся грамматикой из [11].

structured-name ::=

```
    /* Однозначный полный образец */  
    '(' capitalized-ident ')'  
    /* Однозначный частичный образец */  
    | '(' capitalized-ident '|_|)'  
    /* Многозначный полный образец */  
    | '(' capitalized-ident {'|' capitalized-ident} ')'
```

value-name ::= ...

```
| structured-name
```

pattern ::= ...

```
    /* Параметризованный частичный образец */  
    | '<' capitalized-ident expr {expr} '>' pattern
```

Поскольку на данный момент не существует формального определения статической семантики языка OCaml, а также по ряду причин, описанным в исходной работе [10], единственным возможным способом для презентации правил типизации является представление эталонного типизирующего анализатора, реализованного в данной работе [2].

Компиляция в промежуточное представление (1/3)

Мы представим расширение оптимизирующей схемы компиляции [6], используемой в актуальных версиях компилятора OCaml.

Схема дополняется двумя параметрами: *вхождение* o [7] и *кэш-карта* $amap$. В качестве результата теперь схема также дополнительно возвращает *кэш-карту*.

Разделяющее правило дополняется случаем разделения по активному образцу и теперь последовательно передает *кэш-карту*:

$$l_q, \rho_q, amap_q = C^*(\vec{x}, Q \rightarrow M, partial, (R, e); def, ctx, o, amap)$$

$$l_r, \rho_r, amap_r = C^*(\vec{x}, R \rightarrow N, ex \quad , \quad def, ctx, o, amap_q)$$

Компиляция в промежуточное представление (2/3)

Добавляется новое правило для компиляции активных образцов. Входная матрица $P \rightarrow L$ имеет вид $(A(q_1, \dots, q_a) p_2 \dots p_n \rightarrow l)$ и активный образец $(|A|)$ является единичным полным. Выполнив рекурсивный вызов, получим:

$$l_c, \rho, \text{map} = C^*((y_1 \dots y_a x_2 \dots x_n), (q_1 \dots q_a p_2 \dots p_n \rightarrow l), \text{ex}, \Downarrow(\text{def}), \Downarrow(\text{ctx}), o \cdot 0, \text{map})$$

Если $\text{map}(o)$ уже содержит $(|A|)$ в качестве ключа z генерируем:

```
(let (y1 (field z 1))  
    ...  
    (ya (field z a))  
    l_c))
```


Если $\text{map}(o)$ не содержит $(|A|)$ в качестве ключа, то дополнительно генерируем свежее имя z и оборачиваем l_r выше в:

```
(seq (setfield 0 z (apply (|A|) x1)) l_r)
```

Компиляция в промежуточное представление (3/3)

Наконец для начального вызова компиляции сопоставления необходимо произвести генерацию необходимых переменных кэшей. Пусть имеем $l, _, \text{map} = C^*(\dots), (z_1, t_1), \dots, (z_n, t_n)$ список пар всех выделенных переменных и их типов (выведенных из типов сохранённых идентификаторов активных образцов), тогда код l необходимо обернуть в:

```
(let (z_1 = (makemutable 0 (t_1) 0))
      ...
      (z_n = (makemutable 0 (t_n) 0))
  l)
```


- 1 Произведен полный обзор существующих решений для статически типизируемого сопоставления с образцом
 - ▶ По результатам был представлен доклад [15] на [FProg SPB 27.02.2020](#) ;
 - ▶ Произведен отбор критериев и сравнение существующих решений [3];
 - ▶ Утверждено реализуемое решение и сформирован RFC [1];
- 2 Формализация и реализация необходимых изменений:
 - ▶ Описан и реализован синтаксический анализ для расширения;
 - ▶ Реализованы правила типизации;
 - ▶ Описаны (но не реализованы) необходимые изменения схемы компиляции.
- 3 Прототип реализации [2] (без поддержки компиляции)

Библиография и ссылки I



Bashkirov Alexander. Active patterns for OCaml. —
2020. —

Режим доступа: <https://github.com/ocaml/RFCs/pull/12> (дата обращения: 01.06.2020).



Bashkirov Alexander. Active patterns for OCaml implementation repository. —
2020. —

Режим доступа: https://github.com/bash-spbu/ocaml/tree/active_patterns_docker_image (дата обращения: 01.06.2020).

Библиография и ссылки II



Bashkirov Alexander. Rethinking pattern matching as an embedded DSL. —

2020. —

Режим доступа: <https://github.com/bash-spbu/ways-of-pattern-matching-extending-overview/blob/master/text/ways-of-pattern-matching-extension-overview.md> (дата обращения: 01.06.2020).



Emir Burak, Odersky Martin, Williams John. Matching Objects with Patterns // Proceedings of the 21st European Conference on Object-Oriented Programming. —

ECOOP'07. —

Berlin, Heidelberg : Springer-Verlag, 2007. —

С. 273–298. —

Режим доступа:

<http://dl.acm.org/citation.cfm?id=2394758.2394779>.

Библиография и ссылки III



Erwig Martin, Peyton Jones Simon. Pattern Guards and Transformational Patterns // Haskell Workshop 2000. — 2000. — September. —

Режим доступа:

<https://www.microsoft.com/en-us/research/publication/pattern-guards-and-transformational-patterns/>.





Le Fessant Fabrice, Maranget Luc. Optimizing Pattern Matching // ACM SIGPLAN Notices. —

2001. — 08. —

T. 36.

Библиография и ссылки IV

-  Maranget Luc. [Compiling Pattern Matching to Good Decision Trees](#) // Proceedings of the 2008 ACM SIGPLAN Workshop on ML. — ML '08. —
New York, NY, USA : ACM, 2008. —
С. 35–46. —
Режим доступа: <http://doi.acm.org/10.1145/1411304.1411311>.
-  Okasaki Chris. [Views for Standard ML](#) // In SIGPLAN Workshop on ML. —
1998. —
С. 14–23.

Библиография и ссылки V



Pattern Synonyms / Matthew Pickering, Gergo Érdi, Simon Peyton Jones, Richard A. Eisenberg // Haskell'16. — 2016. — September. —

Режим доступа: <https://www.microsoft.com/en-us/research/publication/pattern-synonyms/>.





Syme Don, Neverov Gregory, Margetson James. Extensible pattern matching via a lightweight language extension // Proceedings of the 12th ACM SIGPLAN international conference on Functional programming. —

Association for Computing Machinery, Inc., 2007. — October. —

Режим доступа:

<https://www.microsoft.com/en-us/research/publication/extensible-pattern-matching-via-a-lightweight-language-exte>

Библиография и ссылки VI

-  The OCaml system release 4.09: Documentation and user's manual : Intern report / Inria ; Executor: Xavier Leroy, Damien Doligez, Alain Frisch et al. : 2019. — Sep. — P. 1–789. Access mode: <https://hal.inria.fr/hal-00930213>.
-  Tullsen Mark. First Class Patterns // In 2nd International Workshop on Pracial Aspects of Declarative Languages, volume 1753 of LNCS. — Springer-Verlag, 2000. — C. 1–15.

Библиография и ссылки VII



Wadler P. *Views: A Way for Pattern Matching to Cohabit with Data Abstraction* // Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. — POPL '87. —

New York, NY, USA : ACM, 1987. —

C. 307–313. —

Режим доступа: <http://doi.acm.org/10.1145/41625.41653>.



Wiki GHC. *Musings on extended pattern-matching syntaxes*. — 2017. —

Режим доступа:

<https://gitlab.haskell.org/ghc/ghc/wikis/view-patterns>
(дата обращения: 01.06.2020).



Башкиров Александр. Полный обзор сопоставления с образцом в статически типизируемых языках. — 2020. —

Режим доступа: https://drive.google.com/open?id=15XD3HNm-XLTw9CQuXaDIkIQvtpbl7i_Y (дата обращения: 01.05.2020).