Wprowadzenie do języka.

Porównanie z Prologiem.

Plan wykładu

- Wprowadzenie
- Typy proste
- Operatory
- Typy złożone
- Zmienne
- Dopasowywanie wzorców
- Funkcje i moduły
- Strażnicy
- Listy

Historia

- 1982 - Eksperymenty z programowaniem telekomunikacyjnym za pomocą> 20 różnych języków (Lisp , Prolog , Parlog ...)
- 1985 - Eksperymenty z Lisp, Prolog, Parlog itp. Wnioski: język musi zawierać współbieżność i łatwe usuwanie błędów (wyklucza Lisp i Prolog.) muszą również posiadać granulację współbieżności tak, że jeden asynchronicznych proces telefonii reprezentowany jest przez jeden proces w języku. (wyklucza Parlog.) Zatem trzeba stworzyć własny język z cechami Lisp, Prolog i Parlog, ale z odzyskiem

Zainteresowanie

Kliknij, aby edytować style wzorca tekstu

- Drugi poziom
  - Trzeci poziom
    - Czwarty poziom
      - Piąty poziom

Cechy języka

- Funkcyjny
- Deklaratywny
- Dynamicznie typowany – typy są sprawdzane w trakcie działania programu
- Silnie typowany – brak niejawnych konwersji
- Zmienne pojedynczego przypisania
- Komunikacja międzyprocesowa wbudowana w język
- Obsługa błędów wbudowana w język

Zastosowanie języka

- TAK:

Systemy rozproszone

Systemy współbieżne

Systemy udostępniania usług

Bazy danych

- Nie za bardzo:

GUI

Grafika

Zaczynamy działać

- Kompilator Erlang OTP: http://erlang.org
- Narzędzia oraz maszyna wirtualna.
- Pełna dokumentacja języka, tutoriale.
- Edycja kodu:
- Notepad, gedit, vim itd. , dowolny edytor tekstowy.
- Eclipse – wtyczka erlIDE.
- Książki
- Programming Erlang, Joe Armstrong
- Erlang Programming, F. Cesarini, S. Thompson

Typy proste

Liczby całkowite

Zakres ograniczony jedynie wielkością pamięci.

Małe liczby przechowywane są jako słowa, a duże jako bignum

16#val – system szesnastkowy

$c – kod ASCI

```
Erlang R14B01 (erts-5.8.2) [smp:2:2] [rq:2] [async-threads:0]

Eshell V5.8.2  (abort with ^G)


1> is_integer(10).

true

2> is_integer(-257).

true
```

```
%Prolog

1 ?- integer(10).
true.

2 ?- integer(-257).
true.

3 ?- integer(1234567890).
true.

4 ?- integer(16#abc).
ERROR: Syntax error: Operator expected

4 ?- integer(a).
false.

5 ?- integer($q).
false.
```

Liczby zmiennoprzecinko

Zapisywane na 64bitach, floating point

IEEE 754

Implementacja jest słabo wydajna

```
Erlang R14B01 (erts-5.8.2) [smp:2:2] [rq:2] [async-threads:0]

Eshell V5.8.2  (abort with ^G)


1> is_float(0.1).

true

2> is_float(34.3456).

true

3> is_float(-3456.789000).

true
```

```
%Prolog

1 ?- float(0.1).
true.

2 ?- float(34.3456).
true.

3 ?- float(-3456.789000).
true.
```

Atomy

- Atomy są to stałe literały.
- Zaczynają się z małej litery, składają się z liter, cyfr i [@_]
- Lub dowolny ciąg zapisany w ‘’
- Zabronione jest stosowanie słów kluczowych.

```
%Prolog
1 ?- atom(piwo).
true.

2 ?- atom(toJestDzieje).
true.

3 ?- atom('Norbert').
true.

4 ?- atom(!@#$%^).
true.

5 ?- atom('dowolne znaki !@#$%^&*()_+').
true.
```

```
Erlang R14B01 (erts-5.8.2) [smp:2:2] [rq:2] [async-threads:0]

Eshell V5.8.2  (abort with ^G)
```

```
1>is_atom(piwo).

true
```

Typy złożone

Tuple (zbiory)

Typ złożony o stałej liczbie elementów.

Ustalona kolejność elementów.

Każdy poprawny typ prosty może się zawierać w tuple.

Podobnie w prologu występują wyrażenia złożone.

(a, b, c)

()

(piwo, 12, 345.345, ‘PWSZ’)

(atom(piwo), 1+2, ())

```
Erlang R14B01 (erts-5.8.2) [smp:2:2] [rq:2] [async-threads:0]

Eshell V5.8.2  (abort with ^G)


1> is_tuple({a,b,c}).

true

2> is_tuple({}).

true

3> is_tuple({piwo, 12, 345.345, 'PWSZ'}).
```

Listy

- Typ złożony o zmiennej liczbie elementów
- Ilość elementów ustalana jest dynamicznie.
- Elementem listy może być każde poprawne wyrażenie.

```
Erlang R14B01 (erts-5.8.2) [smp:2:2] [rq:2] [async-threads:0]

Eshell V5.8.2  (abort with ^G)


1>is_list([x, y, z]).

true

2> is_list([1, 12.34, zosia, {a,b,c}]).

true

3> is_list([123, {1, 2, 3}, [ula, ala], pasztet]).
```

```
%Prolog

[x, y, z]

[1, 12.34, zosia, {a, b, c}]

[123, (1, 2, 3), [ula, ala], pasztet]
```

Ciągi znaków

- Brak typu znakowego string.
- Ciągi znaków są zapisywane jako listy znaków $x kodu ASCII.

```
Erlang R14B01 (erts-5.8.2) [smp:2:2] [rq:2] [async-threads:0]

Eshell V5.8.2  (abort with ^G)


1> "HELLO WORLD".

"HELLO WORLD"

2> [$H, $E, $L, $L, $O, $ , $W, $O, $R, $L, $D].

"HELLO WORLD"

3> is_list("Hello world").

true
```

W prologu stringi również prezentowane są jako listy integerów prezentujących kody ASCII.

"I am a HiLog string”

[73,32,97,109,32,97,32,72,105,76,111,103, 32,115,116,114,105,110,103]

Struktury złożone

Przedstawione tuple oraz listy można w sobie zagnieżdżać.

```
Erlang R14B01 (erts-5.8.2) [smp:2:2] [rq:2] [async-threads:0]

Eshell V5.8.2  (abort with ^G)


1> [ 123,   {a, b, c},   [ {1, 2, 3},  {},  ala ]  ].


[123,{a,b,c},[{1,2,3},{},ala]]
```

Prolog także umożliwia zagnieżdżanie różnych typów.

[123, (a, b, c), [(1, 2, 3), (), ala] }

Zmienne

Zmienne

- Każda nazwa zmiennej musi się zaczynać wielką literą.
- Nazwy mogą składać się z liter cyfr oraz [_@].
- _zmienna – wartość tej zmiennej nie będzie wykorzystywana.
- _ - ta zmienna będzie ignorowana.

```
%Prolog
Zmienna

Zmienna_

_zmiennaAnonimowa
```

```
Erlang R14B01 (erts-5.8.2) [smp:2:2] [rq:2] [async-threads:0]

Eshell V5.8.2  (abort with ^G)

1> Zmienna
```

Wartości zmiennych

- Zmienna może być powiązana z danymi lub strukturami danych.
- Zmienne można powiązać z wartością tylko raz.
- Zmienne dzielimy na związane i niezwiązane.

%Prolog

1 ?- Piwo = maBabelki.
Piwo = maBabelki.

2 ?- Piwo = jestJasne.
Piwo = jestJasne.

3 ?- JakiesDane = (123, ala, [1, 2, 3]).
JakiesDane = (123, ala, [1, 2, 3]).

4 ?- _zmiennaAnonimowa = caCoToKomu.
_zmiennaAnonimowa = caCoToKomu.

5 ?- _ = joker.
true.

- Erlang R14B01 (erts-5.8.2) [smp:2:2] [rq:2] [async-threads:0]
- 1> Piwo = maBabelki.
- maBabelki

Dopasowywanie wzorców

Wzorce

- Nadanie zmiennej wartości to przypisanie wzorca.
- Pozwalają na kontrolowanie wykonywania programu.
- Pozwalają na wyciąganie danych ze struktur.

```
%Prolog
1 ?- (A, B) = (pepsi, cola).
A = pepsi,
B = cola.

2 ?- (A, A, B) = (1, 1, 3).
A = 1,
B = 3.

3 ?- (A, A, B) = (1, 2, 3).
false.

4 ?- (osoba, Imie, _, _) = (osoba, "Zbigniew",
"Kraweznik", [(wiek, 23),(adres, krakow, 123)]).

Imie = [90, 98, 105, 103, 110, 105, 101, 119]
```

```
Erlang R14B01 (erts-5.8.2) [smp:2:2] [rq:2] [async-threads:0]

Eshell V5.8.2  (abort with ^G)


1> {A, B} = {pepsi, cola}.

{pepsi,cola}
```

Funkcje

Funkcje

- Programy w Erlangu składają się głównie z funkcji
- Funkcje definiujemy wewnątrz modułów
- Nazwy funkcji i modułów muszą być atomami
- Ciało funkcji składa się z klauzul
- Każda klauzula używa innego wzorca parametrów
- Funkcja zwraca wartość ostatnio wykonanego wyrażenia

Deklaracja i uruchomienie

Funkcje tworzymy w dowolnym edytorze i zapisujemy z rozszerzeniem .erl, nazwa pliku musi być taka sama jak nazwa modułu. Kompilujemy w emulatorze poleceniem c(nazwa_pliku).

```
-module(test).
-export([fac/1]).

fac(0) -> 1;
fac(N) -> N * fac(N-1).
```

Deklaracja nazwy modułu

Deklaracja nazwy funkcji

Klauzule funkcji fac

Silnia w Prologu

```
silnia(N,F) :-
N>0,
N1 is N-1,
silnia(N1,F1),
F is N * F1.
```

```
Erlang R14B01 (erts-5.8.2) [smp:2:2] [rq:2] [async-threads:0]
Eshell V5.8.2  (abort with ^G)

1> c(silnia).
{ok,silnia}
2> silnia:fac(3).
6
3> silnia:fac(20).
2432902008176640000
```

Strażnicy

Guards

- Słowo kluczowe when pozwala nam na określenie dodatkowego dopasowania do wzorca
- Guards, mogą być stosowani w:
- Nagłówkach funkcji
- Opcjach case
- Warunkach if

```
fac(N) when is_integer(N) and (N > 0) ->
N * fac(N-1);
fac(_) ->
1.
```

Listy

Listy

Mogą przechowywać dowolne poprawnie zapisane typy

```
%Prolog

1 ?- [Pierwszy | Reszta] = [1,2,3,4,5].
Pierwszy = 1,
Reszta = [2, 3, 4, 5].

2 ?- [P1, P2 | R] = [1, 2, 3, 4, 5, 6, 7].
P1 = 1,
P2 = 2,
R = [3, 4, 5, 6, 7].

3 ?- [A, B | C] = [1, 2].
A = 1,
B = 2,
C = [].
```

Jak to wygląda

Przykład

```
-module(tut6).
-export([list_max/1]).
list_max([Head|Rest]) ->
   list_max(Rest, Head).
list_max([], Res) ->
    Res;
list_max([Head|Rest], Result_so_far) when
Head > Result_so_far ->
    list_max(Rest, Head);
list_max([Head|Rest], Result_so_far) ->
```

Przykład

```
-module(tut5).
-export([format_temps/1]).
%% Only this function is exported
format_temps([])->                      % No output for an empty list
    ok;
format_temps([City | Rest]) ->
    print_temp(convert_to_celsius(City)),
    format_temps(Rest).
convert_to_celsius({Name, {c, Temp}}) ->  % No conversion needed
    {Name, {c, Temp}};
convert_to_celsius({Name, {f, Temp}}) ->  % Do the conversion
    {Name, {c, (Temp - 32) * 5 / 9}}.
print_temp({Name, {c, Temp}}) ->
    io:format("~-15w ~w c~n", [Name, Temp]).


1> c(tut5).
{ok,tut5}
2> tut5:format_temps([{moscow, {c, -10}}, {cape_town, {f, 70}}, {stockholm, {c, -4}}, {paris, {f, 28}}, {london, {f, 36}}]).
 moscow -10 c
cape_town 21.11111111111111 c
stockholm -4 c
paris -2.2222222222222223 c
london 2.2222222222222223 c
ok
```

Erlang vs Prolog

Porównanie

Erlang:

```
-module(tut4).

-
export([list_length/1])
.

% simple function to
return list length

list_length([]) ->
        0;
```

Podobieństwa:

- Moduły - po jednym na plik.
- Dyrektywy eksport bardzo podobne.
- Kropki na końcach bloku kodu.
- Komentarze, użyć tego samego symbolu.
- Wiele funkcji z klauzul maja takie same nazwy i podobne argumenty.
- Listy pracują w taki sam sposób: [głowa | ogon].
- Zmienne zaczynają się dużymi literami.
- Atomy zaczyna się małymi literami, (atomy są takie same jak symbole Ruby).
- Zmienne Singleton, generowanie ostrzeżeń kompilatora, chyba że są poprzedzone znakiem podkreślenia (_F powyżej).
- Aby zamknąć powłoki używamy "halt".

Różnice:

- Funkcje Erlanga zwracają wartości, w przeciwieństwie do koncepcji "sukcesów i porażek" języka Prolog i przesyłania zwracanych wartości przez argumenty. Prolog stosuje mechanizm unifikacji (nie zwraca w "tradycyjny sposób" wyniku funkcji)

Gdzie zastosowano

- Facebook (Facebook chat backend)
- T-Mobile (advanced call control services)
- Amazon.com (Amazon simple DB)
- YAHOO! (Yachoo! Delicious)
- nk.pl (nTalk)
- Ericsson (Switches, Telecom Servers)
- Klarna (Electronic payment systems)

Dziękuję za uwagę