KLIENS-SZERVER ALAPÚ ERLANG PROGRAMOK TRANSZFORMÁCIÓJA ERLANG OTP SÉMÁRA

Király Roland

kiralyroland@inf.elte.hu

Támogatók:

- GVOP-3.2.2-2004-07-0005/3.0 ELTE IKKK

- Ericsson Hungary

Az előadás tartalma

- **Refaktorálás**
- **Erlang**
- **Erlang kliens-szerver alkalmazások**
- **Erlang OTP**
- **Transzformáció Erlang OTP sémára**

Refaktoring

A Refaktoring transzformációs eljárás, mely során a programok forráskódját változtatjuk meg a funkcionalitás megváltoztatása nélkül.

A refaktorálás elvégzése előtt analizálni kell a kódot, mert a transzformáció nem minden esetben végezhető el.

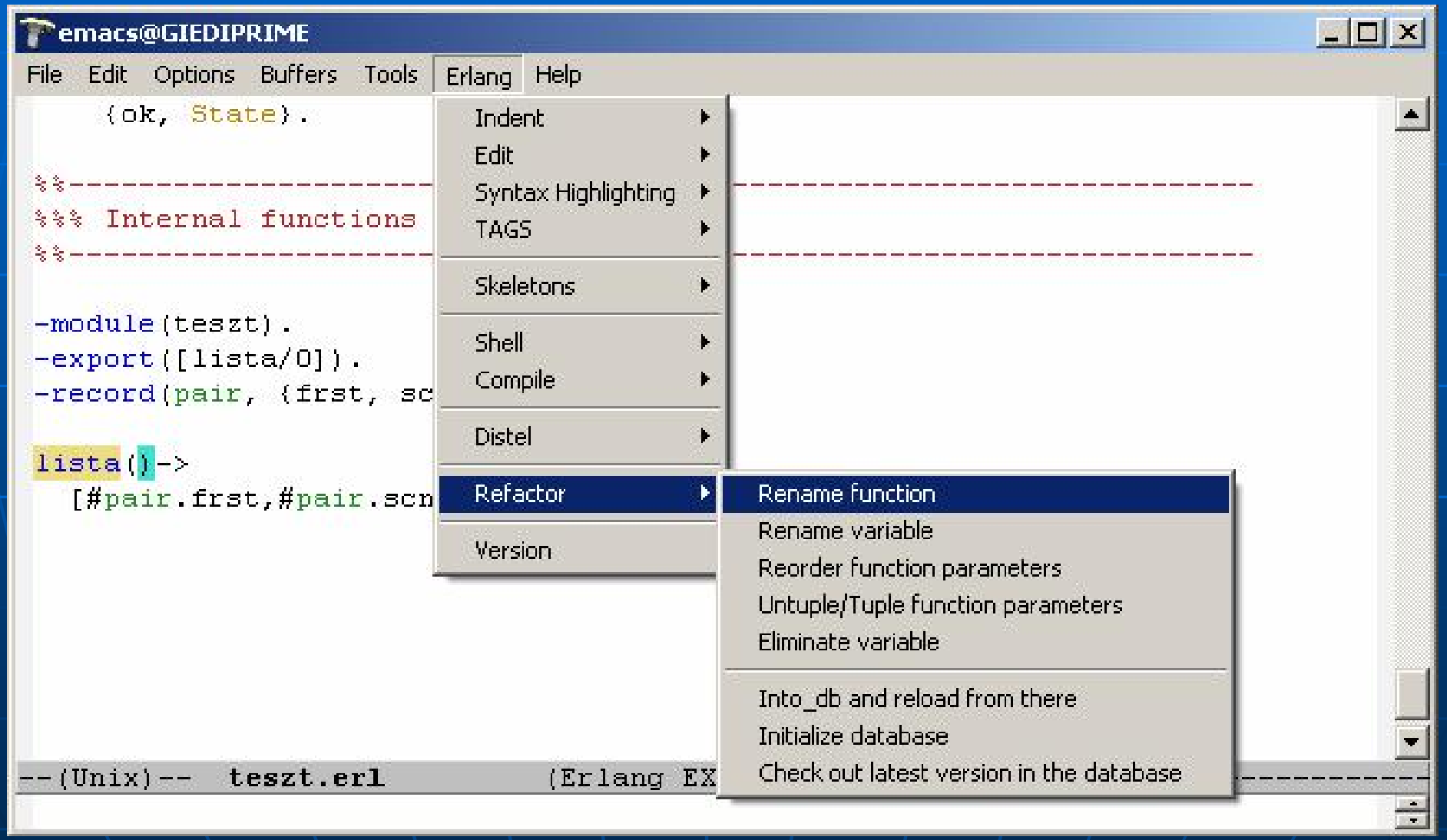
Példa refaktorálásra

Merge subexpression duplicates:

```
-module(refac).
-export([foo/2]).

foo(A,B) ->
    pid ! {note, A+B},
    A+B.
```

```
-module(refac).
-export([foo/2]).

foo(A,B) ->
    V = A+B,
    pid ! {note, V},
    V.
```

Refactoring tool



Az Erlang programozási nyelv

- **Az Erlang funkcionális programozási nyelv, mely alkalmas nagy hibatűrő képességgel rendelkező, elosztott alkalmazások fejlesztésére.**
- **Az Ericsson telekommunikációs rendszerek fejlesztésére használja**
- **Közös memória nélkül, üzenetküldésekkel valósítja meg az elosztottságot**
- **Támogatja más nyelven írt programkomponensek integrálását**
- **Rendkívül gyenge a típusossága**

Erlang elemei

- Atomok, Számok, Szövegek, Változók
- Rendezett n-esek (tuple)
- Listák, Halmaz kifejezések
- Processz ID
- Függvények user/BIF-s
- Magasabb rendű függvények (HOF)
- Mintaillesztés
- Rekurzió
- Üzenetküldés (Message Passing)

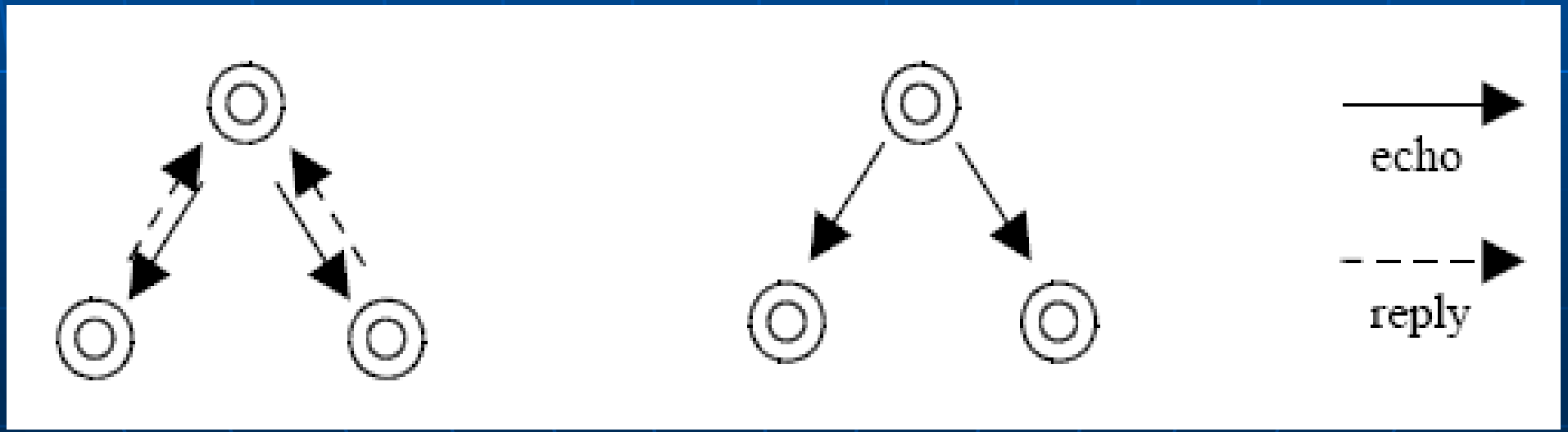
Erlang programok szerkezete

```
-module(modulneve).
-export([fuggveny_pattern/1]).
-import([foo/0]).

fuggveny() -> foo().

fuggveny_pattern(Param) ->
    fuggveny_torzs_1;

fuggvemy_pattern(Param1, Param2) ->
    fuggveny_torzs_2.
```

Erlang folyamatok

Az Erlang programok modulokból állnak. A futó erlang programok olyan folyamatok, amelyek nem használnak közös memóriát.

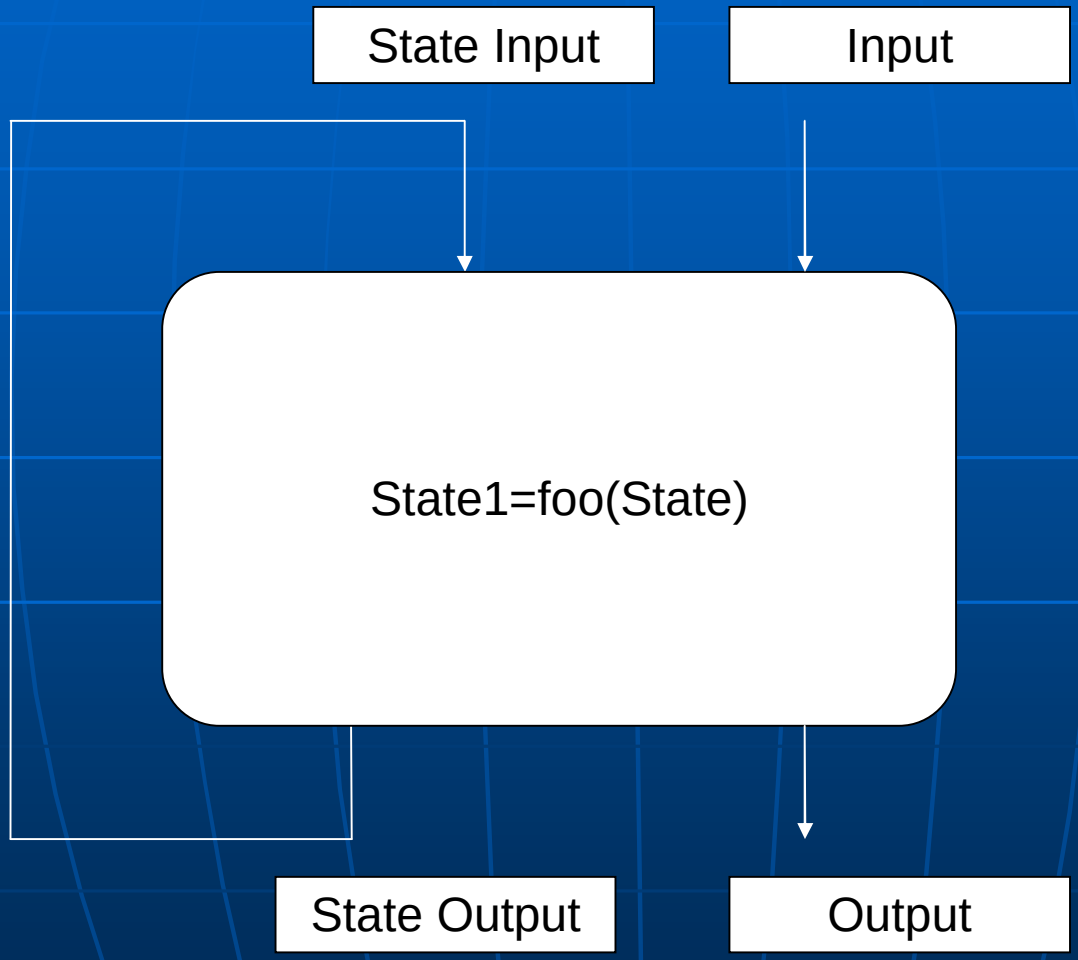
A kommunikációt és az adatcserét üzenetek küldésével oldják meg. Az üzenetküldés lehet szinkron, vagy aszinkron működésű.



Kliens-szerver erlang programok

```
1. -module(server).
2. -export([start/0,init/1]).

3. start() ->
4.    register(server_,spawn(server, init, [])).

5. init(State)-> loop(State).

7. loop(State) ->
8.    receive
9.      {client_1, A} ->
            State1=foo(State, A), loop(State1);
10.     {client_3, C, Pid} ->
11.         Pid ! {server_, ok},
12.         State1=foo(State, C), loop(State1).
13.   end
```

Állapotátvitelek



Kliens-szerver alkalmazások hibái

Az állapotok átvitele (Finite State) rekurzióval oldható meg, mely rossz kód esetén a verem telítődését okozhatja!

A kliens és a szerver kódokat nehéz azonosítani

Kód módosításakor le kell állítani az alkalmazást

A szerver nem írható le általánosan

Rekurziv és Iteratív megoldások

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

Az N*fact(N-1) résszel van a probléma, mivel minden egyes futáskor adatokat kell elhelyezni a verembe.

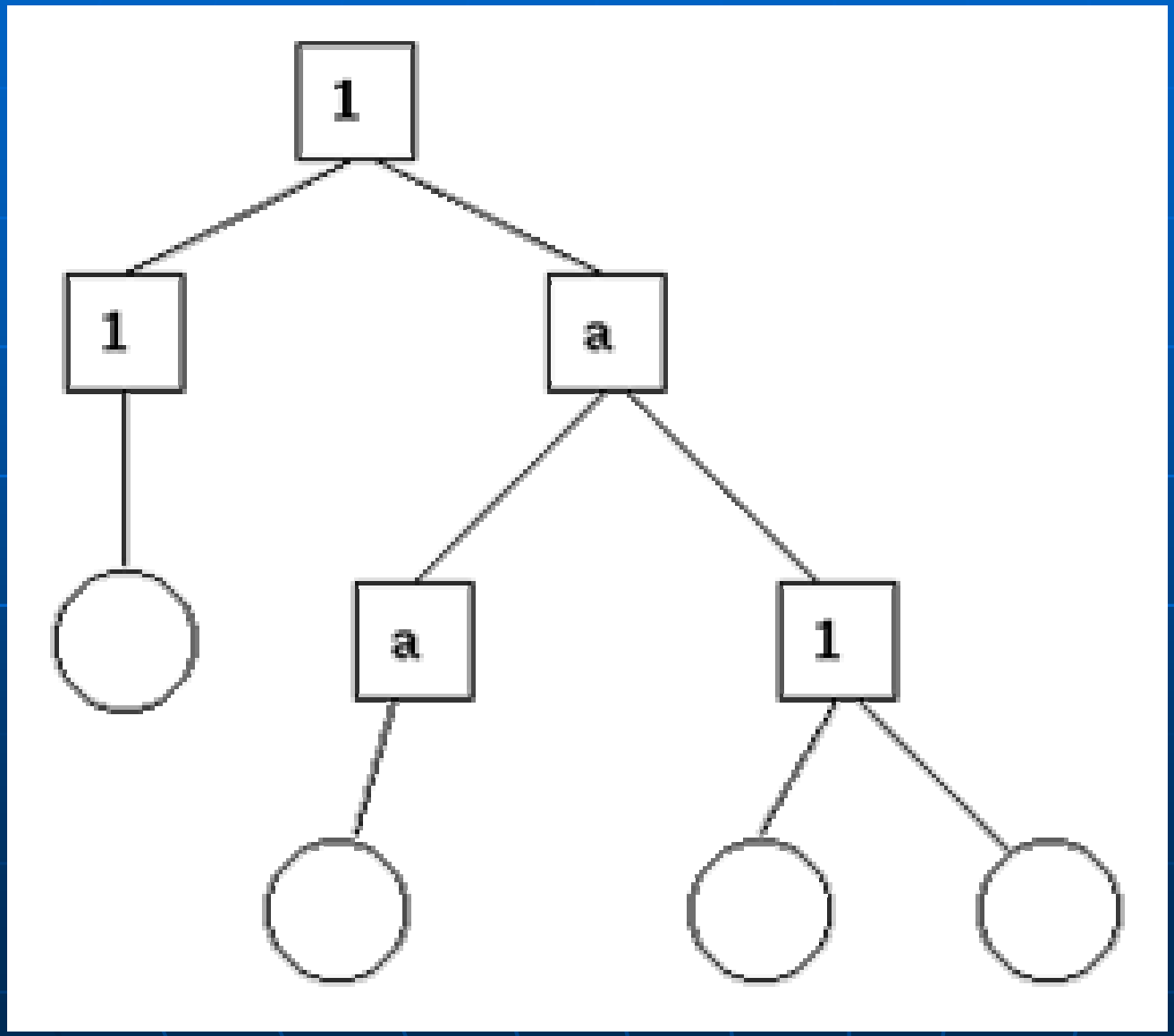
Iteratív megoldás:

```
fact(N) -> fact_x(N, 1).

fact_x(0, X) ->  X;
fact_x(N, X) -> fact(N-1, N*X).
```

Erlang OTP

Open Telecom Platform

Supervisor tree



Supervisor

Worker

Erlang/OTP

- gen_server
- gen_event
- gen_fsm
- supervisor

Generic Server

- **Kliens – szerver folyamatok leírása**
- **Magasabb absztrakciós szintre emeljük a kódot.**
- **Szét tudjuk választani generikus és specifikus részre**
- **A szerverek leírhatóak általánosan – ugyanaz a kód különféle szerverek implementációjára is alkalmas**

Hagyományos szerver kódja

```
1.  -module(server).
2.  -export([start/0,init/1]).

3.  start() ->
4.    register(server_,spawn(server, init, [])).

5.  init(State)-> loop(State).

7.  loop(State) ->
8.    receive
9.      {client_1, A} ->
            State1=foo(State, A), loop(State1);
10.     {client_3, C, Pid} ->
11.         Pid ! {server_, ok},
12.         State1=foo(State, C), loop(State1).
13.   end
```

Szerver callback modul

```
1.  -module(server).
2.  -behaviour(gen_server).

5.  start() ->
6.      gen_server:start_link({local,server}, server, [], []).

7.  init(State) -> {ok, server}.

9.  handle_cast(_Request, State) ->
10.      State1=foo(_Request), {noreply, State1}.

11. handle_call(_Request, _From, State) ->
12.      State1=foo(_Request),{reply, State1, State1}.
```

```
13. client_1() ->
14.      gen_server:cast(server, {client_1,message1}).

15. client_3() ->
16.      gen_server:call(server, {client_3,message3}).
```

Eredmény

```
loop(State) ->
  receive
    {c_1, A} ->
          State1=foo(State, A), loop(State1);
    {c_2, B} ->
          State1=foo(State, B), loop(State1)
  end
```

helyett a kódban:

```
handle_cast(_Request, State) ->
                  State1=foo(_Request).
```

Köszönöm a figyelmet!

Kérdések?