Recovering Erlang AST from BEAM bytecode

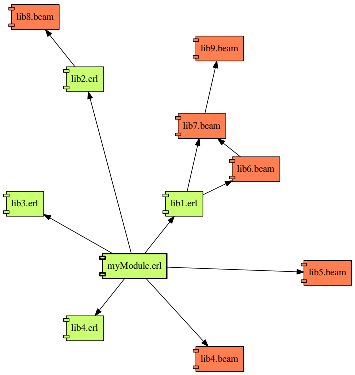
Dániel Lukács and Melinda Tóth

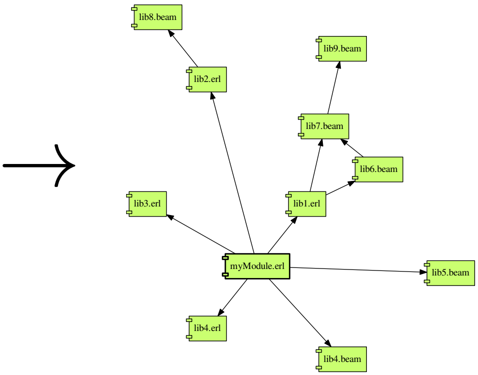
Eötvös Loránd University HU,
Faculty of Informatics

[0] This research has been supported by the Hungarian Government through the New National Excellence Program of the Ministry of Human Capacities.

Motivation

Problem statement

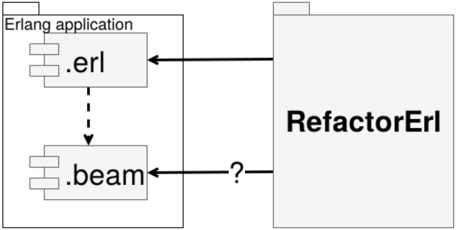




Goal

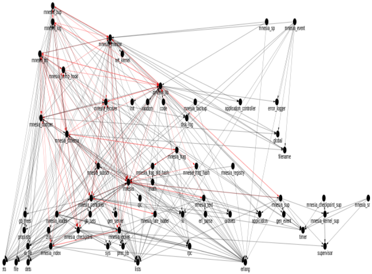
Problem statement

Enable the **RefactorErl** static analysis system for **Erlang** to recover information from source dependencies stored as Erlang **BEAM bytecode**.



RefactorErl

Problem statement

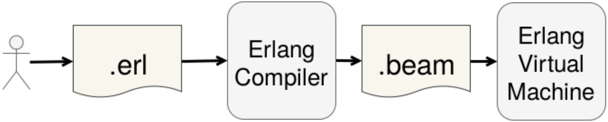


RefactorErl

- Static analysis framework for Erlang
- 2006, ELTE Faculty of Informatics, Department of Programming Languages And Compilers
- Functionality:
  - Semantic queries
  - Refactorings
  - Dependency analysis
  - Code metrics, investigation, duplicated code analysis, and more.
  - Web interface, GUI, CLI
- https://plc.inf.elte.hu/erlang/

RefactorErl

Problem statement




```
%% Rectangular
    function
rect(T)
    when abs(T)  <  1;
         abs(T) =:= 1
    -> 1;

rect(_)
    -> 0.
```

```
{function, rect, 1, 6}.
{label,5}.
{line,[{location,"example.erl",10}]}.
{func_info,{atom,example},{atom,rect
      },1}.

{label,6}.
{gc_bif,abs,{f,7},1,[{x,0}],{x,1}}.
{test,is_ge,{f,8},[{x,1},{integer,1}]}.

{label,7}.
{gc_bif,abs,{f,9},1,[{x,0}],{x,1}}.
{test,is_eq_exact,{f,9},[{x,1},{integer
      ,1}]}.

{label,8}.
{move,{integer,1},{x,0}}.
return.

{label,9}.
{move,{integer,0},{x,0}}.
return.
```

Machine code vs. BEAM

Problem statement

Machine code

- Low-level (CPU)
- Heavily optimized code
- Complicated loop constructs
- Dynamic memory allocation
- Indirect addressing
- Stack frames
- Manual process control
- No metainformation about modules and functions

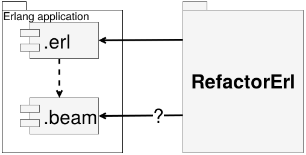
BEAM bytecode

- High-level (VM)
- Simplified code
- Loops always correspond to specific Erlang constructs
- Managed memory allocation with garbage collection
- Registers, index displacement
- Special registers for local variables
- Process scheduling managed by VM
- Stores metainformation about modules and functions

Goal (revision)

Problem statement

Enable the **RefactorErl** static analysis system for **Erlang** to recover information from source dependencies stored as Erlang **BEAM bytecode**.



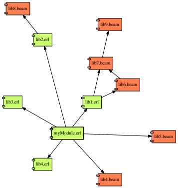
Approach: Represent recovered BEAM semantical information in **Erlang syntax**.

- Ready for RefactorErl
- Results can be compared with the original
- Existing decompilation techniques can be used

What is the problem?

Problem statement

```
-module(event_handler).
-export([handle_event/1]).

handle_event(Event) ->
  case Event of
    ok -> done;
    {message, _} -> done;
    _ -> unknown_event
  end.
```

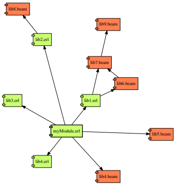


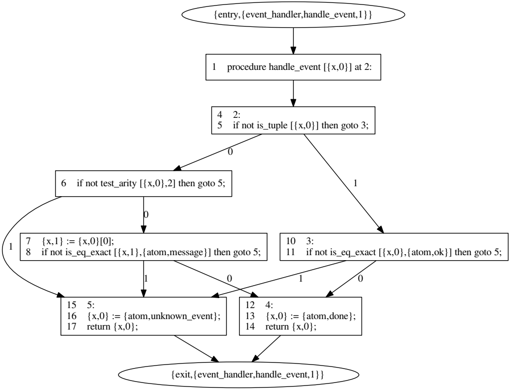

```
{module, event_handler}.  %% version = 0
{exports, [{handle_event,1},{module_info,0},{module_info,1}]}.
{attributes, []}.
{labels, 10}.

{function, handle_event, 1, 2}.
{label,1}.
{line,[{location,"event_handler.erl",4}]}.
{func_info,{atom,event_handler},{atom,handle_event},1}.
{label,2}.
{test,is_tuple,{f,3},[{x,0}]}.
{test,test_arity,{f,5},[{x,0},2]}.
{get_tuple_element,{x,0},0,{x,1}}.
{test,is_eq_exact,{f,5},[{x,1},{atom,message}]}.
{jump,{f,4}}.
{label,3}.
{test,is_eq_exact,{f,5},[{x,0},{atom,ok}]}.
{label,4}.
{move,{atom,done},{x,0}}.
return.
{label,5}.
{move,{atom,unknown_event},{x,0}}.
return.
```

Is it starightforward? NO!

Problem statement

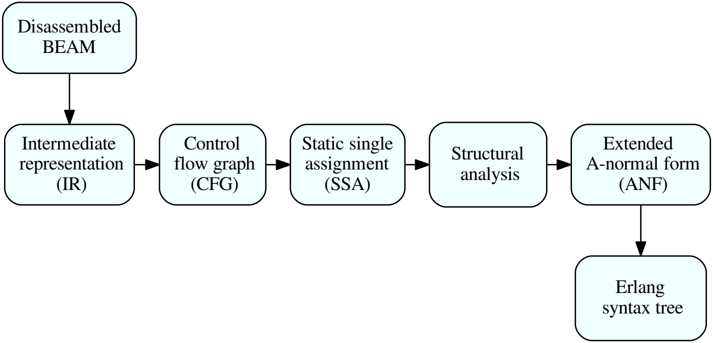
```erlang
-module(event_handler).
-export([handle_event/1]).

handle_event(Event) ->
  case Event of
    ok -> done;
    {message, _} -> done;
    _ -> unknown_event
  end.
```





Decompiler workflow

Methodology



Disassambled BEAM

Methodology

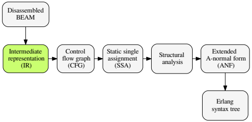
Disassemblers shipped with Erlang/OTP:

- beam_disasm module
- erlc -S

```
{module, event_handler}.  %% version = 0
{exports, [{handle_event,1},{module_info,0},{module_info,1}]}.
{attributes, []}.
{labels, 10}.

{function, handle_event, 1, 2}.
{label,1}.
{line,[{location,"event_handler.erl",4}]}.
{func_info,{atom,event_handler},{atom,handle_event},1}.
{label,2}.
{test,is_tuple,{f,3},[{x,0}]}.
{test,test_arity,{f,5},[{x,0},2]}.
{get_tuple_element,{x,0},0,{x,1}}.
{test,is_eq_exact,{f,5},[{x,1},{atom,message}]}.
{jump,{f,4}}.
{label,3}.
{test,is_eq_exact,{f,5},[{x,0},{atom,ok}]}.
{label,4}.
{move,{atom,done},{x,0}}.
return.
{label,5}.
{move,{atom,unknown_event},{x,0}}.
return.
```

Intermediate Representation

Methodology



- Explicit syntax
- Abstraction
- Internal model

```
{move,{atom,done},{x,0}}.
{x,0} := {atom,done};
```

```
{call_ext_only,1,{extfunc,erlang,get_module_info,1}}.
{x,0} := {atom,event_handler};
{x,0} := call erlang:get_module_info/1 [{x,0}];
```

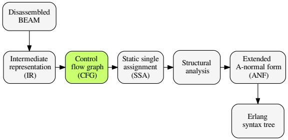
```
procedure handle_event [{x,0}] at 2:
1:
throw {function_clause,{atom,event_handler},{atom,handle_event},1};
2:
if not is_tuple [{x,0}] then goto 3;
if not test_arity [{x,0},2] then goto 5;
{x,1} := {x,0}[0];
if not is_eq_exact [{x,1},{atom,message}] then goto 5;
goto 4;
3:
if not is_eq_exact [{x,0},{atom,ok}] then goto 5;
4:
{x,0} := {atom,done};
return {x,0};
5:
{x,0} := {atom,unknown_event};
return {x,0};
```

An internal model of BEAM semantics.

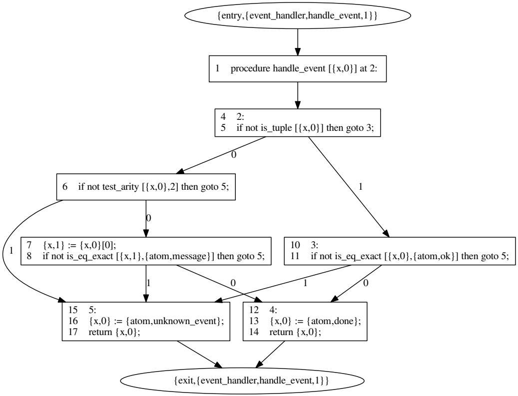
- Some BEAM instructions has **implicit semantics**. Making it explicit reduces possibility for implementation errors.
- **Abstraction layer**
  - BEAM semantics is not formally documented
  - BEAM semantics may change

Control Flow Graph (CFG)

Methodology

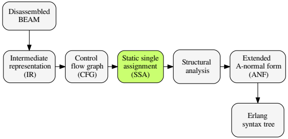


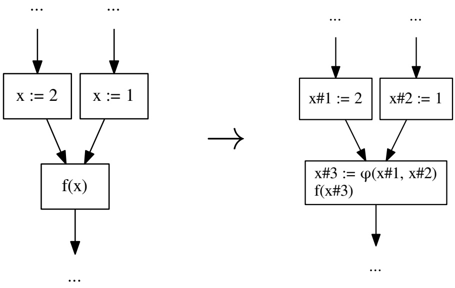
- Equivalent graph representation of the program flow
- Consists of blocks (nodes) and flows (edges).
- Base of further analyses
  - Dominator
  - Static Single Assignment
  - Structuring
- Simple transformations



Static Single Assignment (SSA)[1]

Methodology



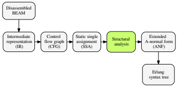


- Each symbol is defined only once
- Calculated based on dominator information
- SSA ↔ Functional representation

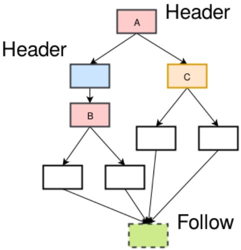
[1] Cytron, Ron; Ferrante, Jeanne; Rosen, Barry K.; Wegman, Mark N. & Zadeck, F. Kenneth (1991), *Efficiently computing static single assignment form and the control dependence graph*, ACM Transactions on Programming Languages and Systems.

Structuring[2]

Methodology



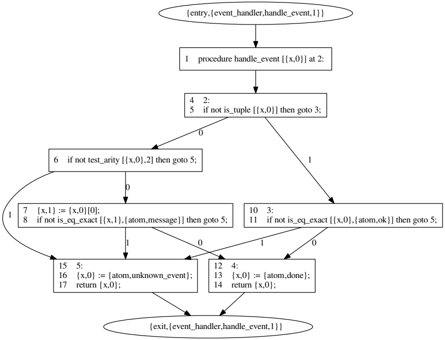
- Identifying high level control structures
- Region: one entry and exit point
- Nested regions



[2] Cifuentes C. (1996), *Structuring decompiled graphs.* In: Gyimóthy T. (eds) Compiler Construction. CC 1996. Lecture Notes in Computer Science, vol 1060. Springer, Berlin, Heidelberg

Structuring[3]

Methodology



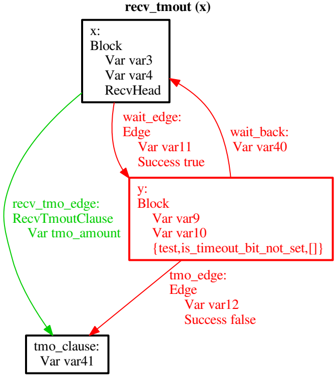
- Unstructured control flow
- goto
- No decomposition
- Pattern based analysis

[3] Cifuentes C. (1996), *Structuring decompiled graphs.* In: Gyimóthy T. (eds) Compiler Construction. CC 1996. Lecture Notes in Computer Science, vol 1060. Springer, Berlin, Heidelberg

Structuring[5]

Methodology

- Graph Rewriting (TGR)[4]
- Formal semantics and provable properties
- Expressive, visualisable

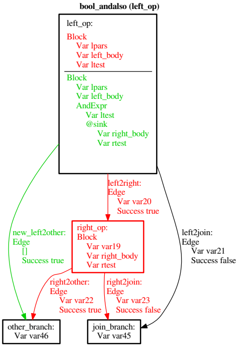


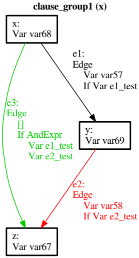
[4] Ehrig, Hartmut and Pfender, Michael and Schneider, Hans-Jürgen. Graph-grammars: An algebraic approach", Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on, pp. 167-180, 1973, IEEE

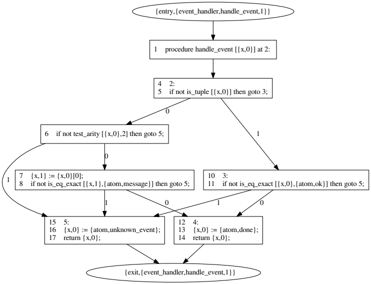
Structuring[6]

Methodology

Erlang branching structures

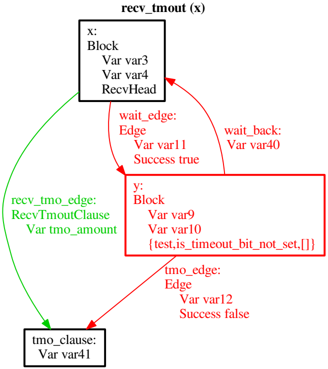


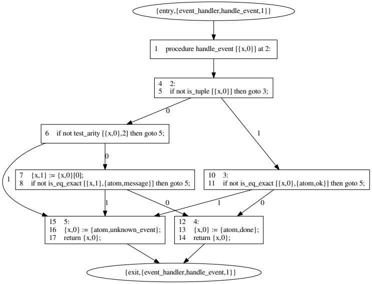




6 Cifuentes C. (1996), *Structuring decompiled graphs.* In: Gyimóthy T. (eds) Compiler Construction. CC 1996. Lecture Notes in Computer Science, vol 1060. Springer, Berlin, Heidelberg

Context-analysis



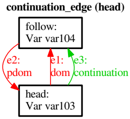


[7] Cifuentes C. (1996), *Structuring decompiled graphs.* In: Gyimóthy T. (eds) Compiler Construction. CC 1996. Lecture Notes in Computer Science, vol 1060. Springer, Berlin, Heidelberg

Structuring[8]

Methodology

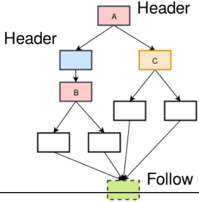
Identifying regions

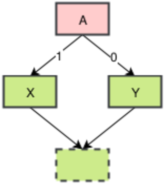


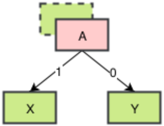

```
if A
then (f X1)
else (f X2)
```

→

```
let
X3 =
   if A
   then X1
   else X2
in
(f X3)
```



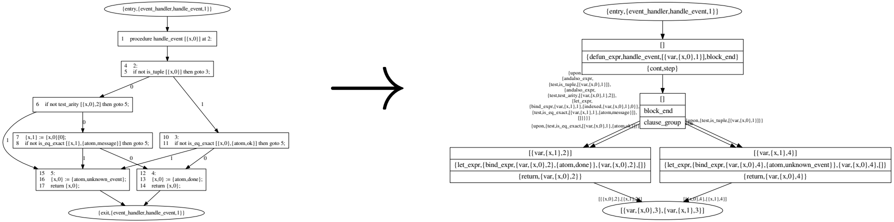




[8] Cifuentes C. (1996), *Structuring decompiled graphs.* In: Gyimóthy T. (eds) Compiler Construction. CC 1996. Lecture Notes in Computer Science, vol 1060. Springer, Berlin, Heidelberg

Structuring[9]

Methodology

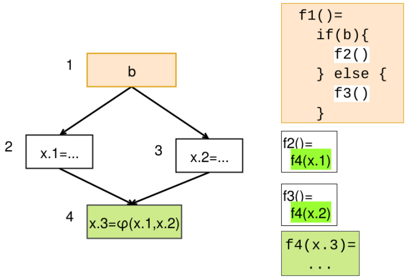


[9] Cifuentes C. (1996), *Structuring decompiled graphs.* In: Gyimóthy T. (eds) Compiler Construction. CC 1996. Lecture Notes in Computer Science, vol 1060. Springer, Berlin, Heidelberg

A-Normal Form

Methodology

- Functional-style program representation
- Translates out of SSA[10]

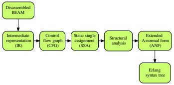



```
FUN handle_event [x0#1] =
  IF
    WHEN
      ANDALSO
        {test,is_tuple,[x0#1]}
        ANDALSO
          {test,test_arity,[x0#1,2]}
          LET
            x1#1 = {indexed,x0#1,0}
          IN
          {test,is_eq_exact,[x1#1,{atom,message}]} ->
      LET
        x0#2 = {atom,done}
      IN      x0#2

    WHEN
      {test,is_eq_exact,[x0#1,{atom,ok}]} ->
      LET
        x0#2 = {atom,done}
      IN      x0#2

    WHEN true ->
      LET
        x0#4 = {atom,unknown_event}
      IN      x0#4
```

[10] Manuel M.T. Chakravarty, Gabriele Keller, and Patryk Zadarnowski (2004), *A Functional Perspective on SSA Optimisation Algorithms.* Electronic Notes in Theoretical Computer Science, Volume 82, Issue 2, April 2004, Pages 347-361

Code generation

Methodology




```
-module(event_handler).
-export([handle_event/1]).

handle_event(X0_1) ->
    if
        (is_tuple(X0_1) andalso (size(X0_1) =:= 2 andalso
        element(1, X0_1) =:= message)) ->
            X0_2 = done,
            X0_2;
        X0_1 =:= ok ->
            X0_2 = done,
            X0_2;
        true ->
            X0_4 = unknown_event,
            X0_4
    end.
```

```
-module(event_handler).
-export([handle_event/1]).

handle_event(Event) ->
  case Event of
    ok -> done;
    {message, _} -> done;
    _ -> unknown_event
  end.
```

DEMO

Module	Funs	Blocks	Total	GR	Referl	Others
mnesia_event	15	165	51.08 s	22.81 s	27.20 s	1.0734 s
mnesia_text	23	166	120.6614 s	75.67 s	43.81 s	1.1814 s
mnesia_index	46	325	347.23 s	232.48 s	112.81 s	1.9280 s

Future Notes

- Pre-structuring
  - List-comprehension
  - Fun-expression
- Full Erlang coverage
  - Catch patterns
  - Binaries
- Extended language support (Elixir)