Chorer v1.1

Gabriele Genovese

March 21, 2024

Overview

Introduction

4 main parts:

- Db initialization
- Metadata extraction
- Generation of the localviews
- Generation of the globalview

```
generate(InputFile, EntryPoint, OutDir) ->
    io:fwrite("Analysing ~p, entrypoint: ~p~n", [InputFile, EntryPoint]),
    Settings = #setting{output_dir = OutDir},
    init_db(),
    md:extract(InputFile),
    lv:generate(Settings),
    gv:generate(Settings, EntryPoint).
```

Figure: `chorer.erl`'s function

Initialization

- Initialize all the ets tables.

```
init_db() ->
    ets:new(?CLINE, [set, named_table]),
    ets:new(?DBMANAGER, [set, named_table]),
    ets:new(?FUNAST, [set, named_table]),
    ets:new(?LOCALVIEW, [set, named_table]),
    ets:new(?REGISTERDB, [set, named_table]),
    ets:new(?ARGUMENTS, [set, named_table]),
    ets:new(?SPAWNC, [set, named_table]).
```

Figure: In `chorer.erl` file

Metadata extraction

- Takes all the function AST and possible actors.

```
%%% Parse all the file and save the Ast of each function and also all the possible actors.
%%% All the exported functions are cosidereted possible actors.
gen_fun_ast_and_exported(Ast) ->
    ActorList =
        lists:foldl(
            fun(CodeLine, AccActorList) ->
                case CodeLine of
                    {attribute, _, export, AtrList} ->
                        AccActorList ++ [share:merge_fun_ar(N, A) || {N, A} <- AtrList];
                    {function, Line, Name, Arity, FunAst} ->
                        % io:fwrite("[MD] Found ~p~n", [share:merge_fun_ar(Name, Arity)]),
                        ets:insert(?FUNAST, {
                            share:merge_fun_ar(Name, Arity), {function, Line, FunAst}
                        }),
                        AccActorList;
                    _ ->
                        AccActorList
                end
            end,
            [],
            Ast
        ),
    ets:insert(?DBMANAGER, {?ACTORLIST, ActorList}).
```

Figure: In `choreography/md.erl` file

Localview

- Generate a localview for each actor.

```
-record(localview, {
    fun_name = "",
    fun_ast = {},
    graph = digraph:new(),
    min_graph = digraph:new(),
    last_vertex = 1,
    local_vars = [],
    ret_var = #variable{},
    edge_map = #{},
    settings = #setting{}
}).
```

Figure: Localview record

- Evaluation of the AST in the `eval` module.

Eval module

- evaluation of the program for lv module
- the output of every function is a lv record

```
%%% API
-export([
    function_list/2,
    clause/5,
    match/3,
    case_pm/3,
    if_pm/2,
    receive_pm/2,
    operation/4,
    function_call/3,
    anon_function/3,
    simple_type/3,
    atom/2,
    list/3,
    map/2,
    tuple/2,
    variable/2
]).
```

Eval module example: `function_call`

```
%%% @doc
%%% Evaluate the call of a function.
function_call(Function, ArgList, Data) ->
    case Function of
        {atom, _, Name} -> call_by_atom(Name, ArgList, Data);
        {var, _, VarName} -> call_by_var(VarName, ArgList, Data);
        {remote, _, Package, FunName} -> call_by_package(Package, FunName, ArgList, Data);
        F -> share:warning("couldn't recognize function call pattern", F, Data)
    end.
```

Figure: Function call API

```
call_by_atom(Name, ArgList, Data) ->
    FunName = Data#localview.fun_name,
    RealName = share:ltoa(share:remove_last(share:remove_last(FunName))),
    case Name of
        RealName -> recursive(ArgList, Data);
        spawn -> spawn_call(ArgList, Data);
        spawn_monitor -> spawn_monitor_call(ArgList, Data);
        self -> self_call(Data);
        register -> register_call(ArgList, Data);
        _ -> generic_call(Name, ArgList, Data)
    end.
```

Figure: Subfunction of a function call

Globalview

- Generate a globalview starting from the `entrypoint` given in input.
- A globalview is a graph (from the `digraph` module).
- Simulation of an actor in the `actor_emul` module.

How does it work

- A branch is a possible execution of the program.

```
progress(BranchList) ->
  list:foreach(
    fun(B) ->
      progress_branch(B)
    end,
    BranchList
  ).
```

```
-record(branch, {
    graph = digraph:new(),
    last_vertex = 1,
    proc_pid_m = #{},
    states_m = #{}
}).
```

Branching

When does a branch duplicate and takes another path?

- When there's a fork in a localview
- When there are possible processes receive
- When evaluating possible message receive within a process

```
progress_branch(BranchData) ->
    % this may generate new branches
     progress_each_proc_until_recv(),
    % this may generate new branches
     evaluate_possible_recv_match().
```

Actor simulation

- An actor is simulated from the `actor_emul.erl` module.

```
proc_loop(Data) ->
   receive
      {use_transition, E} -> ...
      {get_data} -> ...,
      {add_message, M} -> ...,
      ...
   end.
```

```
-record(actor_info, {
    fun_name = "",
    id = ?UNDEFINED,
    current_state = 1,
    first_marked_edges = [],
    second_marked_edges = [],
    spawn_vars = sets:new(),
    local_vars = sets:new(),
    message_queue = []
}).
```

Changelog

Main changes:

1. Refactor of the localview module, partial refactor of the globalview module
2. Implemented parsing of anonymous function, multiple function definition and match of a tuple
3. Github pages with `ex_doc` documentation (automated with Github Action)
4. Added a README for each example
5. Better naming convention for processes `Name/Arity.Seq`
6. Minimization of each lv and gv is asked to the user
7. Added lots of warning

TODO list

A not-exhaustive list of things to do (random order):

- full list support as exchange data (send)
- map support
- bool support
- match with list support
- good attribute passing system when there's a function call
- basic math and logic operations
- order evaluation of receive
- better duplicate on gv
- consider register dynamic
- check the behaviour of the examples and fix bugs
- consider spawn in gv dynamically
- tool for autonomous tests
- write type definition for record and functions

How to contribute