Advanced Programming

Exam 2018

Exam Number: 95, Username: zlp432

November 9, 2018

Contents

1 Utility functions

The Code for this task is attached in the appendix A.1.

1.1 Version

The Implementation of Version is relatively straight forward and throughly tested by unit tests, which include the examples from the exam text. I did ended up with a not working implementation before, so I ended up reimplementing the function which is now working as it should.

1.2 Merge

Merge is implemented as described in the exam text and tested with many different examples in the unit tests, which all run through. I had some problems with matching the constraints together, since I kind of lost overview of the function. Especially ending up when merging only with same package and the different ones (not matching) where not added to the resulting list but in the end just forgot to append the rest to the result.

1.3 Assessment

The Utility functions seems to work as intended, at least I was able to reuse them in the parser, and thanks to lots of unit tests to both functions I do believe they work as they should.

2 Parsing appm databases

2.1 Choice of parser library

I implemented the Parser for appm in parsec, mostly out of this reason:

- Better Error handling compared to ReadP
- I do have more experience with Parsec then ReadP (Assignments)

I did end up using **try** quite a lot, which wasn't my intention at all but with the presented Grammar I haven't found a better solution and overall the parser works more or less.

2.2 Grammar

I decided to make a more strict choice about the Clauses, by parsing them in a fixed ordering (name first etc.), I didn't find much of a better solution for that grammar.

2.3 Assessment

I did quite a few unit tests for the parser (including failing ones), since not everything ended up to be working or there was just not enough time left to fixing all the bugs which showed up.

3 Solver

4 Earls of Ravnica

The code for this task can be found in Appendix

4.1 Solution

4.2 Implementation

The earls of Ravnica can be seen as a state machine for which I chose to use gen_statem. The following states exist:

- Under Configuration
- Under Activation
- Active
- Shutting down

4.3 Data Structure

The Data structure I used to implement Ravnica consists of a map with following entries:

- **description** Saves the description which gets saved when starting a server
- **connections** Map for Handling the connections from one District to an other
- **creatures** Map for handling all the entered/active creatures on a Server
- **trigger** Set a trigger for a district

4.4 All states

Messages which get accepted in all states.

4.4.1 get_description

Gets the description `Desc` which gets set on create of a District.

4.5 Under configuration

As soon as a Server started it is in the under_configuration state.

4.5.1 connect

Connects 2 District with a Action, by saving it in the `connections` map, connects can only be made while district is under configuration in other states an error gets returned.

4.5.2 trigger

Under configuration also a trigger can be added to the server, here always the last one gets taken (overwritting whit the newest one). Trigger gets rung whenever a creature enters or leaves a district.

4.6 Under activation

When `activate` gets called the district and it's neighbors need to get activated, under_activation is a intermediate state until all neighbors and the district itself are activated. In case the neighbors can't be activated (for example when a neighbor got shutdown), then the server goes back to the state of under_configuration.

4.6.1 activate

Activate tries to activate all it's neighbors and changes the state of the server to `active` or back to `under_configuration`.

4.7 Active

In the active state, no more new connections can be added, also no triggers. So as soon as a district and it's neighbors is activated, it should only be possible to either run `get_description`, `enter` or `take_action` and of course shutting down.

4.8 Shutting down

When shutting down is called all neighbors of a district will be shut down as well and this can be propagated until all districts and it's nieghbors are shutdown.

4.8.1 shutdown

4.9 Territories with cycle

A Code Listing

A.1 Question 1.1: handin/appm/src/Utils.hs

```haskell
module Utils where

-- Any auxiliary code to be shared by Parser, Solver, or tests
-- should be placed here.

import Defs

instance Ord Version where
    (<=) (V []) (V []) = True
    (<=) (V ((VN _ _):_)) (V []) = False
    (<=) (V []) (V ((VN _ _):_)) = True
    (<=) (V ((VN v1int v1str) : vnmbr1)) (V ((VN v2int v2str) : vnmbr2))
        | v1int < v2int = True
        | v1int > v2int = False
        | length(v1str) < length(v2str) = True
        | length(v1str) > length(v2str) = False
        | v1str < v2str = True
        | v1str > v2str = False
        | otherwise = (V vnmbr1) <= (V vnmbr2)

merge :: Constrs -> Constrs -> Maybe Constrs
merge [] [] = Just []
merge c1 [] = Just c1
merge [] c2 = Just c2
merge (const:c1) (c2) = case constInC2 const c2 [] of
                            Just x -> merge c1 (x)
                            Nothing -> Nothing

-- Check if Constraint from c1 is in the Constraint list C2
constInC2 :: (PName, PConstr) -> Constrs -> Constrs -> Maybe Constrs
constInC2 const [] x = Just (x ++ [const])
constInC2 const (c2const:c2tail) x =
                case fst const == fst c2const of
                    True -> case mergeConst (snd const) (snd c2const) of
```

```
35                                  Nothing -> Nothing
36                                  Just mconst -> Just (x ++ [(fst const,
                                    ↪  mconst)] ++ c2tail)
37                      False -> constInC2 const c2tail (x ++ [c2const])
38
39  -- Compare the 2 Constraints with
40  mergeConst :: PConstr -> PConstr -> Maybe PConstr
41  mergeConst (b1,c1v1,c1v2) (b2,c2v1,c2v2)
42          | c1v2 <= c2v1 = Nothing
43          | c2v2 <= c1v1 = Nothing
44          | b1 == True && b2 == True = Just (b1, (largest c1v1 c2v1),
            ↪  (smallest c1v2 c2v2))
45          | b1 == False && b2 == False = Just (b1, (largest c1v1 c2v1),
            ↪  (smallest c1v2 c2v2))
46          | b1 == True && b2 == False = Just (b1, (largest c1v1 c2v1),
            ↪  (smallest c1v2 c2v2))
47          | b1 == False && b2 == True = Just (b2, (largest c1v1 c2v1),
            ↪  (smallest c1v2 c2v2))
48  mergeConst _ _ = Nothing
49
50  -- Return the smaller of 2 Versions
51  smallest :: Version -> Version -> Version
52  smallest v1 v2 =
53      case v1 <= v2 of
54          True -> v1
55          False -> v2
56
57  -- Returns the bigger of 2 Versions
58  largest :: Version -> Version -> Version
59  largest v1 v2 =
60      case v1 >= v2 of
61          True -> v1
62          False -> v2
```

A.2 Question 2.1: handin/ravnica/district.erl

```
1   -module(district).
2   -behaviour(gen_statem).
3   -export([create/1,
4     get_description/1,
5     connect/3,
6     activate/1,
7     options/1,
8     enter/2,
9     take_action/3,
10    shutdown/2,
11    trigger/2]).
```

```erlang
%% Gen_statem callbacks
-export([terminate/3, code_change/4, init/1, callback_mode/0]).
%State Functions
-export([under_configuration/3, active/3, shutting_down/3,
    under_activation/3]).
-type passage() :: pid().
-type creature_ref() :: reference().
-type creature_stats() :: map().
-type creature() :: {creature_ref(), creature_stats()}.
-type trigger() :: fun((entering | leaving, creature(), [creature()])
  -> {creature(), [creature()]}).


-spec create(string()) -> {ok, passage()} | {error, any()}.
create(Desc) ->
  gen_statem:start(?MODULE, Desc, []).

-spec get_description(passage()) -> {ok, string()} | {error, any()}.
get_description(District) ->
  gen_statem:call(District, get_description).

-spec connect(passage(), atom(), passage()) -> ok | {error, any()}.
connect(From, Action, To) ->
  gen_statem:call(From, {connect, Action, To}).

-spec activate(passage()) -> active | under_activation | impossible.
activate(District) ->
  gen_statem:call(District, activate).

-spec options(passage()) -> {ok, [atom()]} | none.
options(District) ->
  gen_statem:call(District, options).

-spec enter(passage(), creature()) -> ok | {error, any()}.
enter(District, Creature) ->
  gen_statem:call(District, {enter, Creature}).

-spec take_action(passage(), creature_ref(), atom()) -> {ok, passage()} |
    {error, any()}.
take_action(From, CRef, Action) ->
  gen_statem:call(From, {take_action, CRef, Action}).

-spec shutdown(passage(), pid()) -> ok.
shutdown(District, NextPlane) ->
  gen_statem:call(District, {shutdown, NextPlane}).

-spec trigger(passage(), trigger()) -> ok | {error, any()} | not_supported.
trigger(District, Trigger) ->
  gen_statem:call(District, {trigger, Trigger}).
```

```
59
60
61  %% States
62  handle_event({call, From}, get_description, Data) ->
63    case maps:is_key(description, Data) of
64      true -> {keep_state, Data, {reply, From, {ok, maps:get(description,
        ↪  Data)}}};
65      false -> {error, "No Description"}
66    end;
67
68  handle_event({call, From}, options, Data) ->
69    {keep_state, Data, {reply, From, {ok, maps:keys(maps:get(connections,
      ↪  Data))}}};
70
71  % ignore all other unhandled events
72  handle_event({call, From}, activate, Data) ->
73    {next_state, active, Data, {reply, From, ok}};
74
75  handle_event({call, From}, {run_action, CRef, Stats}, Data) ->
76    case maps:is_key(CRef, maps:get(creatures, Data)) of
77      true -> {keep_state, Data, {reply, From, {error, "Creature is already in
        ↪  this District"}}};
78      false -> NewCreatures = maps:put(CRef, Stats, maps:get(creatures,
        ↪  Data)),
79        NewData = maps:update(creatures, NewCreatures, Data),
80        {keep_state, NewData, {reply, From, ok}}
81    end;
82
83  % Handle Enter on other states
84  handle_event({call, From}, {enter, _}, Data) ->
85    {keep_state, Data, {reply, From, {error, "Can't enter in this state"}}};
86
87  % Shutdown can be called in any state
88  handle_event({call, From}, {shutdown, NextPlane}, Data) ->
89    NextPlane ! {shutting_down, From, maps:to_list(maps:get(creatures,
      ↪  Data))},
90    {next_state, shutting_down, Data, {next_event, internal, {From,
      ↪  NextPlane}}};
91
92  handle_event({call, From}, {trigger, _Trigger}, Data) ->
93    {keep_state, Data, {reply, From, {error, "Can't set a trigger in this
      ↪  state"}}};
94
95  handle_event({call, From}, {connect, _Action, _To}, Data) ->
96    {keep_state, Data, {reply, From, {error, "Can't connect in this state"}}};
97
98  % ignore all other unhandled events
99  handle_event(_EventType, _EventContent, Data) ->
100   {keep_state, Data}.
```

```
101
102 under_configuration({call, From}, {connect, Action, To}, Data) ->
103   case is_process_alive(To) of
104     true -> case maps:is_key(Action, maps:get(connections, Data)) of
105               false -> Connections = maps:put(Action, To,
                  ↪  maps:get(connections, Data)),
106                 NewData = maps:update(connections, Connections, Data),
107                 {keep_state, NewData, {reply, From, ok}};
108               true -> {keep_state, Data, {reply, From, {error, "Action
                  ↪  already exists"}}}
109             end;
110     false -> {keep_state, Data, {reply, From, {error, "Process not alive
        ↪  anymore"}}}
111   end;
112
113 under_configuration({call, From}, activate, Data) ->
114   {next_state, under_activation, Data, {next_event, internal, From}};
115
116
117 under_configuration({call, From}, {trigger, Trigger}, Data) ->
118   NewData = maps:update(trigger, Trigger, Data),
119   {keep_state, NewData, {reply, From, ok}};
120
121 %% General Event Handling for state under_configuration
122 under_configuration(EventType, EventContent, Data) ->
123   handle_event(EventType, EventContent, Data).
124
125 under_activation(internal, From, Data) ->
126   Result = broadcast_connection(maps:to_list(maps:get(connections, Data)),
      ↪  From, active),
127   case Result of
128     impossible -> {next_state, under_configuration, Data, {reply, From,
        ↪  Result}};
129     active -> {next_state, active, Data, {reply, From, Result}}
130   end;
131
132 under_activation({call, From}, activate, Data) ->
133   {keep_state, Data, {reply, From, under_activation}};
134
135 under_activation({call, From}, options, Data) ->
136   {keep_state, Data, {reply, From, {ok, maps:keys(maps:get(connections,
      ↪  Data))}}};
137
138 %% General Event Handling for state under_activation
139 under_activation(EventType, EventContent, Data) ->
140   handle_event(EventType, EventContent, Data).
141
142 active({call, From}, {enter, {Ref, Stats}}, Data) ->
143   case maps:is_key(Ref, maps:get(creatures, Data)) of
```

```
144       true -> {keep_state, Data, {reply, From, {error, "Creture is already in
          ↪  this District"}}};
145       false -> Creatures = maps:get(creatures, Data),
146         case maps:get(trigger, Data) of
147           none -> Creature1 = none, Creatures1 = none;
148           Trigger -> case run_trigger(Trigger, entering, {Ref, Stats},
              ↪  Creatures) of
149                        {error, _} -> Creature1 = none, Creatures1 = none;
150                        {Creature1, Creatures1} -> {Creature1, Creatures1}
151                      end
152         end,
153         case {Creature1, Creatures1} of
154           {none, none} -> NewCreatures = maps:put(Ref, Stats,
              ↪  maps:get(creatures, Data)),
155             NewData = maps:update(creatures, NewCreatures, Data);
156           {{Ref1, Stats1}, NewCreatures1} -> NewCreatures = maps:put(Ref1,
              ↪  Stats1, maps:from_list(NewCreatures1)),
157             NewData = maps:update(creatures, NewCreatures, Data)
158         end,
159         {keep_state, NewData, {reply, From, ok}}
160     end;
161
162   active({call, From}, {take_action, CRef, Action}, Data) ->
163     case maps:is_key(Action, maps:get(connections, Data)) of
164       true ->
165         case maps:is_key(CRef, maps:get(creatures, Data)) of
166           false -> {keep_state, Data, {reply, From, {error, "Creature doesn't
              ↪  exist in this district"}}};
167           true -> case maps:get(trigger, Data) of
168                     none -> Creature1 = none, Creatures1 = none;
169                     Trigger ->
170                       RemoveCreature = maps:remove(CRef, maps:get(creatures,
                          ↪  Data)),
171                       RemovedData = maps:update(creatures, RemoveCreature,
                          ↪  Data),
172                       case run_trigger(Trigger, leaving, {CRef, maps:get(CRef,
                          ↪  maps:get(creatures, Data))},
173                         maps:get(creatures, RemovedData)) of
174                         {error, _} -> Creature1 = none, Creatures1 = none;
175                         {Creature1, Creatures1} -> {Creature1, Creatures1}
176                       end
177                   end,
178             case {Creature1, Creatures1} of
179               {none, none} -> NewDataCreatures = Data;
180               {{Ref, Stats}, _} -> NewCreatures = maps:put(Ref, Stats,
                  ↪  maps:get(creatures, Data)),
181                 NewDataCreatures = maps:update(creatures, NewCreatures, Data)
182             end,
```

```
183             {NewData, To} = creature_leave(CRef, Action, From,
                ↪  NewDataCreatures),
184             case NewData of
185               error -> {keep_state, Data, {reply, From, {error, To}}};
186               _ -> {keep_state, NewData, {reply, From, {ok, To}}}
187             end
188         end;
189       false -> {keep_state, Data, {reply, From, {error, "Action doesn't
          ↪  exist"}}}
190     end;
191
192   active({call, From}, activate, Data) ->
193     {keep_state, Data, {reply, From, active}};
194
195   %% Handle Calls to active
196   active(EventType, EventContent, Data) ->
197     handle_event(EventType, EventContent, Data).
198
199   shutting_down(internal, {From, NextPlane}, Data) ->
200     Result = broadcast_shutdown(maps:to_list(maps:get(connections, Data)),
      ↪  From, NextPlane),
201     {stop_and_reply, normal, {reply, From, Result}};
202
203   shutting_down({call, From}, activate, Data) ->
204     {keep_state, Data, {reply, From, impossible}};
205
206   shutting_down({call, From}, options, Data) ->
207     {keep_state, Data, {reply, From, none}};
208
209   shutting_down({call, From}, shutdown, Data) ->
210     {keep_state, Data, {reply, From, ok}};
211
212   %% Handle Calls to shutting_down
213   shutting_down(EventType, EventContent, Data) ->
214     handle_event(EventType, EventContent, Data).
215
216   %% Mandatory callback functions
217   terminate(_Reason, _State, _Data) ->
218     void.
219
220   code_change(_Vsn, State, Data, _Extra) ->
221     {ok, State, Data}.
222
223   % initial State under_configuration
224   init(Desc) ->
225     %% Set the initial state + data
226     State = under_configuration, Data = #{description => Desc, connections =>
      ↪  #{}, creatures => #{}, trigger => none},
227     {ok, State, Data}.
```

```
228
229 callback_mode() -> state_functions.
230
231 %% Synchronous Call which should wait until each response
232 broadcast_shutdown([], _, _NextPlane) -> ok;
233 broadcast_shutdown([{_Action, To} | Actions], {Pid, Ref}, NextPlane) ->
234   case is_process_alive(To) of
235     true ->
236       case term_to_binary(To) == term_to_binary(Pid) of
237         true -> void;
238         false -> case term_to_binary(To) == term_to_binary(self()) of
239                    true -> void;
240                    false -> gen_statem:call(To, {shutdown, NextPlane})
241                  end
242       end;
243     false -> void
244   end,
245   broadcast_shutdown(Actions, {Pid, Ref}, NextPlane).
246
247 %% Synchronous Call which should wait until each response
248 broadcast_connection([], _, Result) -> Result;
249 broadcast_connection([{_Action, To} | Actions], {Pid, Ref}, _) ->
250   case is_process_alive(To) of
251     false -> Result1 = impossible;
252     true -> Result1 = active,
253       case term_to_binary(To) == term_to_binary(Pid) of
254         false -> case term_to_binary(To) == term_to_binary(self()) of
255                    true -> void;
256                    false -> gen_statem:call(To, activate)
257                  end;
258         true -> void
259       end
260   end,
261   broadcast_connection(Actions, {Pid, Ref}, Result1).
262
263 creature_leave(CRef, Action, {_Pid, _}, Data) ->
264   To = maps:get(Action, maps:get(connections, Data)),
265   Stats = maps:get(CRef, maps:get(creatures, Data)),
266   case is_process_alive(To) of
267     true -> case term_to_binary(self()) == term_to_binary(To) of
268               true -> {Data, To};
269               false -> case gen_statem:call(To, {run_action, CRef, Stats})
                  ↪  of
270                          ok -> NewCreatures = maps:remove(CRef,
                             ↪  maps:get(creatures, Data)),
271                            NewData = maps:update(creatures, NewCreatures,
                               ↪  Data),
272                            {NewData, To};
273                          {error, Reason} -> {error, Reason}
```

```
274                         end
275                 end;
276       false -> {error, "District is shutdown"}
277     end.
278
279 run_trigger(Trigger, Event, Creature, Creatures) ->
280     Self = self(),
281     spawn(fun() -> Self ! {self(), Trigger(Event, Creature,
        ↪  maps:to_list(Creatures))} end),
282     receive
283       {_Pid, {Creature1, Creatures1}} -> {Creature1, Creatures1}
284     after
285       2000 -> {error, "didnt't run function"}
286     end.
```

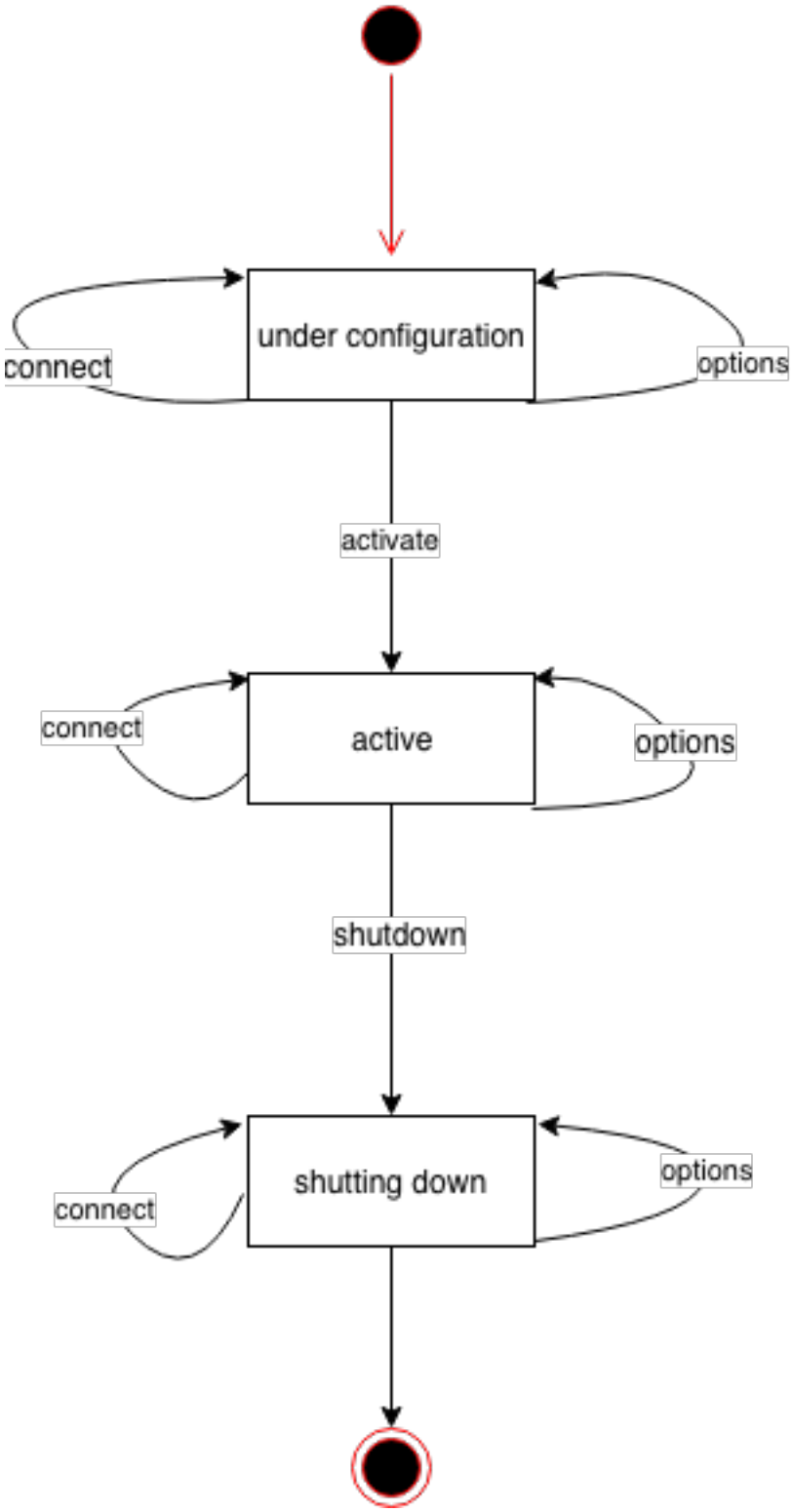

Figure 1: Simple State machine diagramm