

Advanced Programming Exam 2018

Exam Number: 95, Username: zlp432

November 9, 2018

Contents

1	Utility functions	2
1.1	Version	2
1.2	Merge	2
1.3	Assessment	2
2	Parsing appm databases	2
2.1	Choice of parser library	2
2.2	Transforming Grammar	3
2.3	Assessment	4
2.3.1	Scope of Test Cases	4
2.3.2	Correctness	4
2.3.3	Code Quality	4
3	Solving appm constraints	4
4	Testing appm properties	4
5	The district module	5
5.1	Implementation	5
5.2	Data Structure	5
5.3	States	6
5.4	Communication	6
5.5	Cycles	6
5.6	Triggers	7
5.7	Assessment	7
5.7.1	Scope of Test Cases	7
5.7.2	Correctness	8
5.7.3	Code Quality	8
6	QuickCheck district	8
6.1	Territory Generator	8

A	Code Listing	8
A.1	Question 1.1: handin/appm/src/Utils.hs	8
A.2	Question 1.2: handin/appm/src/ParserImpl.hs	10
A.3	Question 2.1: handin/ravnica/district.erl	14
B	Tests Listing	22
B.1	Question 1.1 and Question 1.2 handin/appm/tests/BB/Main.hs	22
B.2	Question 2.1	28
B.3	Question 2.2	34

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 thoroughly 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, as 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

The Code for this task is attached in the appendix A.2.

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 Transforming 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 (making it more dynamic and of course also reduced the complexity of the parser). The existing grammar has some ambiguities (could have had many names, description etc.), for that case I transformed the Grammar a little so that it matches more to my idea. So that Name comes first (at least once) and then one after the other follows or is set to empty. Technically Version, Description etc. could show up more then once but those cases are not handled in the Parser

```

1 Database ::= \epsilon
2           | Package Database
3 Package ::= 'package' '{' Clauses '}'
4 Clauses ::= \epsilon
5           | Clause
6           | Clauses ';' Clauses
7 Clause ::= 'name' PName Version
8 Version ::= \epsilon
9           | 'version' Version
10          | Description
11 Description ::= \epsilon
12              | 'description' String
13              | Constraints
14 Constraints ::= \epsilon
15               | 'requires' PList
16               | 'conflicts'
17               ↪ PList
18
19 PList ::= PItem
20         | PList ',' PItem
21 PItem ::= PName
22         | PName '>=' Version
23         | PName '<' Version
24 Version ::= (see Text)
25 PName ::= (see Text)
26 String ::= (see Text)

```

Since I don't know what exactly will follow after a Name I use try in all the following calls of parsing one of the clauses (Version etc.), this way I can

skip one or the other except for the name.

2.3 Assessment

2.3.1 Scope of Test Cases

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 fix all the bugs which showed up. Overall I tried to test as many cases as possible, which I think I succeeded in because I did find some bugs throughout the development process thanks to the testing.

2.3.2 Correctness

The Solution is far from perfect, but it was able to parse the example intro to the preferred outcome. There are a few bugs on of which is the ordering of requires and conflicts, so in case I want to parse **requires bar ;= 3, bar ; 2**, this would end in an error since the case of having the greater then version in the first place won't work all the other cases work. Also adding a comment **-comment** after the end of the last package doesn't work. Beside those bugs, there are of course also other ways to get the parser to fail. For example only by changing the ordering of clauses and lots of other things. So for a well-formed package (ordering is right, only one clause of name, version etc.) the parser works, in lots of other cases the parser might fail.

2.3.3 Code Quality

I do believe the code is more or less easy to read, I tried to put the things together which belong together which should help understanding it easier. But nonetheless I'm not satisfied with how many tries I needed to solve the task and also the left over bugs which I wasn't able to fix until the end of the exam. On the other hand I'm also quite happy that was I able to put some more functionality into the parser as I thought beforehand I would do, like the string parsing, caseString (keywords can have any case) etc.

3 Solving appm constraints

skipped since time ran out

4 Testing appm properties

skipped since time ran out

5 The district module

The code for this task can be found in Appendix A.3

5.1 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** The initial state of every district, connections can be created, the description called and so on
- **Under Activation** As soon as someone calls **activate**, the server will switch to this intermediate state until all it's neighbors (connections) are active
- **Active** Server is active and Players can enter, take_actions and so on
- **Shutting down** Intermediate state before being fully shut down, since all neighbors also have to be shutdown

So a District Server can go through following states:

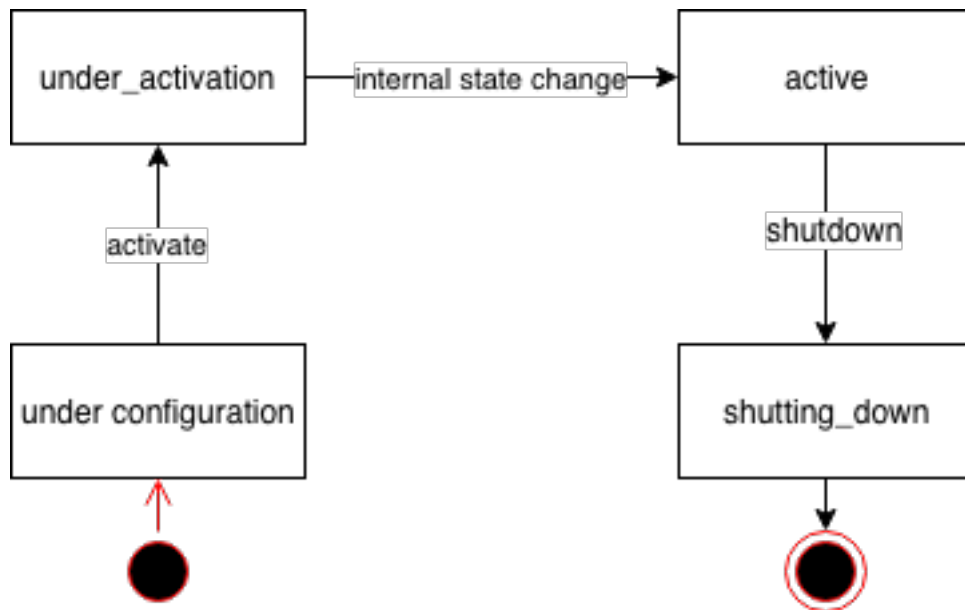


Figure 1: Simple State machine diagramm

5.2 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, which gets called when a creature is leaving or entering a district

5.3 States

In the end I ended up with the described states in which a district can be in. Each state only accepts a number of messages (for example, enter is not allowed in some states and so on), all unhandled Messages (Technically can send any kind of message when the PID is known) will be ignored but the server will be keep going. A maybe new thing I did for the exam is using a internal `next_state` state change, to be able to switch to those intermediate states when needed.

5.4 Communication

All communication between the districts and to the district is synchronous, this mostly for actually know what kind of state the other neighbors have and I do want to know when every neighbors shutting down or that there is an issue with that.

5.5 Cycles

My solution can handle cycles, but it's a very basic solution which definitely won't hold for all kind of cycles. Even though I did a test on cycles (self cycles and cycles with neighbors), which was succesfully. In the end my Solution is about checking the **From** and **To** PID in case they are the same don't end the message further otherwise we end up in a never ending loop. Same for when the PID matches with the own PID of the server (self cycles), **self()** and **To** are the same.

So this very basic solution will fail for bigger cycles, because here the shutdown or activate message won't be sent from the original district (on which the activate or shutdown got called), so can't be checked according to the **To** and **From** PID anymore.

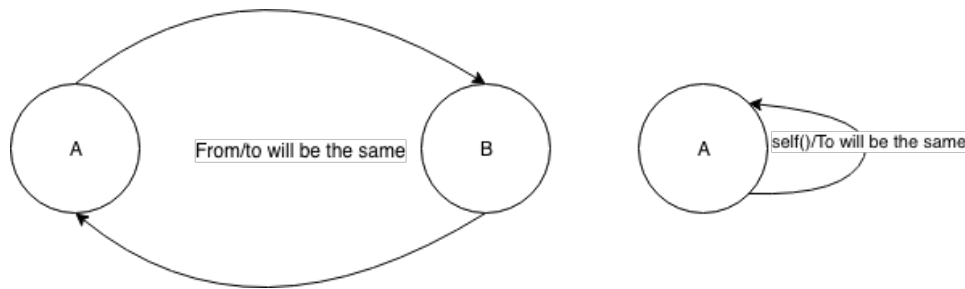


Figure 2: Showcasing the cycles

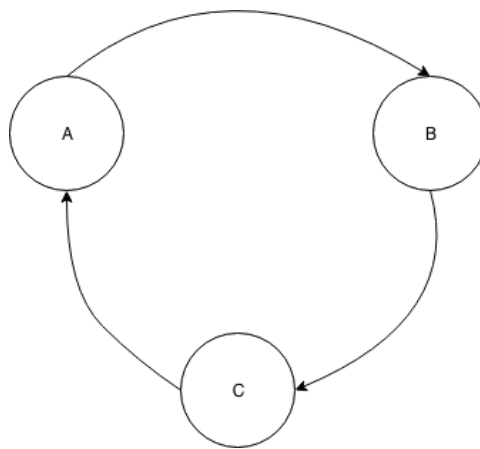


Figure 3: Example of a failing cycle

5.6 Triggers

My Solution does support triggers and according to my tests they seem to work. Also my solution tries to check if a Trigger is well behaved by running it in a different process on which I can set a timeout of 2 seconds, so the server won't break down when a not well behaved trigger runs. I do check that the Trigger needs to have 3 parameters, so technically any function can be passed with 3 parameters but only if the trigger return the right format which is:

```
1 {Creature,Creatures}
```

5.7 Assessament

5.7.1 Scope of Test Cases

I did lot of Test cases with **eunit** which should showcase how much my solution is able to do. So from Cycles to every call possible (at least in the

easiest way), since I don't tested the responses in intermediate states and lots of other things which are not very easy to test just like that.

5.7.2 Correctness

According to my tests I do believe I have a somewhat correct solution, even though the nasty cycles are not in total solved/removed. Otherwise I'm quite confident in my solution and the correctness of it, even though there are probably lots of more nasty test cases I wasn't able to think of in the shot period of time I had.

5.7.3 Code Quality

The code quality is somewhat mediocre, since there is lots of repetition and duplicated code which could be removed/improved but out of the time constraints refactoring wasn't my top priority. Therefore the overall code is ok but not really great, at least all helper functions are grouped together, and the states are ordered by how the server will go through them from top to bottom.

6 QuickCheck district

The code for this task can be found in Appendix B.3

6.1 Territory Generator

I did write a Generator for **territory/0** but I wasn't able to write the Properties for `activate` and `take_action`, since the time ran out. So I only started on this task but wasn't able to finish it totally.

A Code Listing

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

```
1  module Utils where
2
3  -- Any auxiliary code to be shared by Parser, Solver, or tests
4  -- should be placed here.
5
6  import Defs
7
8  instance Ord Version where
9      (<=) (V []) (V []) = True
10     (<=) (V ((VN _ _):_)) (V []) = False
11     (<=) (V []) (V ((VN _ _):_)) = True
```



```

12      (<=) (V ((VN vlint v1str) : vnmb1)) (V ((VN v2int v2str) : vnmb2))
13      | vlint < v2int = True
14      | vlint > v2int = False
15      | length(v1str) < length(v2str) = True
16      | length(v1str) > length(v2str) = False
17      | v1str < v2str = True
18      | v1str > v2str = False
19      | otherwise = (V vnmb1) <= (V vnmb2)
20
21 merge :: Constrs -> Constrs -> Maybe Constrs
22 merge [] [] = Just []
23 merge c1 [] = Just c1
24 merge [] c2 = Just c2
25 merge (const:c1) (c2) = case constInC2 const c2 [] of
26     Just x -> merge c1 (x)
27     Nothing -> Nothing
28
29 -- Check if Constraint from c1 is in the Constraint list C2
30 constInC2 :: (PName, PConstr) -> Constrs -> Constrs -> Maybe Constrs
31 constInC2 const [] x = Just (x ++ [const])
32 constInC2 const (c2const:c2tail) x =
33     case fst const == fst c2const of
34         True -> case mergeConst (snd const) (snd c2const) of
35             Nothing -> Nothing
36             Just mconst -> Just (x ++ [(fst const,
37                 ↪ mconst)] ++ c2tail)
38         False -> constInC2 const c2tail (x ++ [c2const])
39
40 -- Compare the 2 Constraints with
41 mergeConst :: PConstr -> PConstr -> Maybe PConstr
42 mergeConst (b1,c1v1,c1v2) (b2,c2v1,c2v2)
43     | c1v2 <= c2v1 = Nothing
44     | c2v2 <= c1v1 = Nothing
45     | b1 == True && b2 == True = Just (b1, (largest c1v1 c2v1),
46         ↪ (smallest c1v2 c2v2))
47     | b1 == False && b2 == False = Just (b1, (largest c1v1 c2v1),
48         ↪ (smallest c1v2 c2v2))
49     | b1 == True && b2 == False = Just (b1, (largest c1v1 c2v1),
50         ↪ (smallest c1v2 c2v2))
51     | b1 == False && b2 == True = Just (b2, (largest c1v1 c2v1),
52         ↪ (smallest c1v2 c2v2))
53     _ _ = Nothing
54
55 -- Return the smaller of 2 Versions
56 smallest :: Version -> Version -> Version
57 smallest v1 v2 =
58     case v1 <= v2 of
59         True -> v1
60         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 1.2: handin/appm/src/ParserImpl.hs

```

1  module ParserImpl where
2
3  -- put your parser in this file. Do not change the types of the following
4  -- exported functions
5  import Data.Char
6  import Defs
7  import Text.Parsec.Char
8  import Text.Parsec.Combinator
9  import Text.Parsec.Prim
10 import Text.Parsec.String
11 import Utils
12 import Control.Monad (guard)
13
14 parseVersion :: String -> Either ErrMsg Version
15 parseVersion str =
16     case parse
17         (do res <- (many parseVersionN)
18          return res)
19         "Parse Error"
20         str of
21     Left a -> Left (show a)
22     Right b -> Right ((V b))
23
24 parseVersionN :: Parser VNum
25 parseVersionN = do
26     number <- read <$> (many1 (satisfy isDigit))
27     -- Number has to be lower then 1 M
28     guard (number < 1000000)
29     string <- many lower
30     -- Not more then 4 lowercase characters
31     guard (length(string) <= 4)
32     _ <- optional (char '.')
33     return (VN number string)
34
35 parseDatabase :: String -> Either ErrMsg Database
36 parseDatabase db =
37     case parse

```

```

38         (do res <- (many parsePackage)
39             eof
40             return res)
41         "Parse Error"
42     db of
43     Left a -> Left (show a)
44     Right b -> Right (DB b)
45
46 -- Parse Packages
47 parsePackage :: Parser Pkg
48 parsePackage = do
49     _ <- parseWhitespace (caseString "package")
50     _ <- parseWhitespace (string "{")
51     pname <- parseName
52     version <- try parseStringVersion <|> return (V [VN 1 ""])
53     description <- try parseDescription <|> return ""
54     deps <- many (choice [try parseRequires, try parseConflicts])
55     _ <- parseWhitespace (string "}")
56     return
57         Pkg
58         { name = pname
59         , ver = version
60         , desc = description
61         -- filter self referential Constraints
62         , deps = filter (\(name, _) -> name /= pname) (cleanConst (concat
63             ↪ (deps)))
64         }
65
66 -- Parse Package name
67 parseName :: Parser PName
68 parseName = do
69     _ <- parseWhitespace (caseString "name")
70     _ <- parseWhitespace (optional (char '"'))
71     name <- many1 (letter <|> digit <|> char '-' <|> try parseHighComma)
72     guard((last name) /= '-')
73     _ <- optional (char '"')
74     _ <- optional (string ";")
75     return (P name)
76
77 parseStringVersion :: Parser Version
78 parseStringVersion = do
79     _ <- parseWhitespace (caseString "version")
80     version <- parseWhitespace (many1 (digit <|> letter <|> char '.'))
81     optional (string ";")
82     case parseVersion version of
83     Right a -> return a
84     _ -> fail "Version wasn't possible to parse"
85
86 parseDescription :: Parser String

```

```

86 parseDescription = do
87   _ <- parseWhitespace (caseString "description")
88   parseWhitespace (char '"')
89   description <- many (character <|> ( try parseHighComma2))
90   char '"'
91   _ <- optional (string ";")
92   return $ concat(description)
93
94 parseRequires :: Parser Constrs
95 parseRequires = do
96   _ <- parseWhitespace (caseString "requires")
97   pconsts <-
98     parseWhitespace
99     (many
100      (choice
101       [ try (parsePConstrH (True))
102         , try (parseSConstrL (True))
103         , try (parseSConstrH (True))
104       ]))
105   _ <- optional (string ";")
106   return (concat (pconsts))
107
108 parseConflicts :: Parser Constrs
109 parseConflicts = do
110   _ <- parseWhitespace (caseString "conflicts")
111   pconsts <-
112     parseWhitespace
113     (many
114      (choice
115       [ try (parsePConstrH (False))
116         , try (parseSConstrL (False))
117         , try (parseSConstrH (False))
118       ]))
119   _ <- (optional (string ";"))
120   return (concat (pconsts))
121
122 parseSConstrL :: Bool -> Parser Constrs
123 parseSConstrL req = do
124   name <- many1 letter
125   version <- parseWhitespace (parseVersionLow)
126   return [(P name), (req, minV, version)]
127
128 parseSConstrH :: Bool -> Parser Constrs
129 parseSConstrH req = do
130   name <- many1 letter
131   version <- parseWhitespace (parseVersionHigh)
132   return [(P name), (req, version, maxV)]
133
134 parsePConstrH :: Bool -> Parser Constrs

```

```

135 parsePConstrH req = do
136   name <- many1 letter
137   lower <- parseWhitespace (parseVersionLow)
138   _ <- parseWhitespace (string ",")
139   name2 <- parseWhitespace (many1 letter)
140   max <- parseWhitespace (parseVersionHigh)
141   case lower <= max of
142     True ->
143       return [(P name), (req, lower, maxV)], [(P name2), (req, minV, max)]
144     False -> fail "Error"
145
146 parseVersionLow :: Parser Version
147 parseVersionLow = do
148   _ <- string "<"
149   version <- parseWhitespace (many1 (digit <|> letter <|> char '.'))
150   case parseVersion version of
151     Right a -> return a
152     _ -> fail "Version wasn't possible to parse"
153
154 parseVersionHigh :: Parser Version
155 parseVersionHigh = do
156   _ <- string ">="
157   version <- parseWhitespace (many1 (digit <|> letter <|> char '.'))
158   case parseVersion version of
159     Right a -> return a
160     _ -> fail "Version wasn't possible to parse"
161
162 -- Merges parsed constraints to remove duplicates etc.
163 cleanConst :: Constrs -> Constrs
164 cleanConst [] = []
165 cleanConst (x:xs) =
166   case merge xs [x] of
167     Nothing -> []
168     Just a -> a
169
170 -- parsing escape and non escape characters
171 character :: Parser String
172 character = fmap return nonEscape <|> escape
173
174 escape :: Parser String
175 escape = do
176   d <- char '\\'
177   c <- oneOf "\\\"0\n\r\v\t\b\f" -- all the characters which can be escaped
178   return [d, c]
179
180 nonEscape :: Parser Char
181 nonEscape = noneOf "\\\"0\n\r\v\t\b\f"
182
183 -- parses a Comment, starting with --

```

```

184 parseComment :: Parser ()
185 parseComment = do
186     _ <- string "--"
187     _ <- manyTill anyChar (newLine <|> eof)
188     return ()
189
190 --makes newline be of type ()
191 newLine :: Parser ()
192 newLine = do
193     _ <- newline
194     return ()
195
196 parseWhitespace :: Parser a -> Parser a
197 parseWhitespace input = do
198     spaces
199     optional parseComment
200     spaces
201     input
202
203 -- Match any case of the characters
204 caseString :: String -> Parser String
205 caseString s = try (mapM caseChar s) <?> "\"" ++ s ++ "\""
206 caseChar :: Char -> Parser Char
207 caseChar c = char (toLower c) <|> char (toUpper c)
208
209 -- Accept 2 "" return "
210 parseHighComma :: Parser Char
211 parseHighComma = do
212     _ <- char '"'
213     _ <- char '"'
214     return '"'
215
216 -- for the sake of using
217 parseHighComma2 :: Parser [Char]
218 parseHighComma2 = do
219     _ <- char '"'
220     _ <- char '"'
221     return ['"']

```

A.3 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]).
12  %% Gen_statem callbacks
13  -export([terminate/3, code_change/4, init/1, callback_mode/0]).
14  %State Functions
15  -export([under_configuration/3, active/3, shutting_down/3,
16    ↪ under_activation/3]).
17  -type passage() :: pid().
18  -type creature_ref() :: reference().
19  -type creature_stats() :: map().
20  -type creature() :: {creature_ref(), creature_stats()}.
21  -type trigger() :: fun((entering | leaving, creature(), [creature()]
22    ↪ {creature(), [creature()]})
23
24  -spec create(string()) -> {ok, passage()} | {error, any()}.
25  create(Desc) ->
26    gen_statem:start(?MODULE, Desc, []).
27
28  -spec get_description(passage()) -> {ok, string()} | {error, any()}.
29  get_description(District) ->
30    gen_statem:call(District, get_description).
31
32  -spec connect(passage(), atom(), passage()) -> ok | {error, any()}.
33  connect(From, Action, To) ->
34    gen_statem:call(From, {connect, Action, To}).
35
36  -spec activate(passage()) -> active | under_activation | impossible.
37  activate(District) ->
38    gen_statem:call(District, activate).
39
40  -spec options(passage()) -> {ok, [atom()]} | none.
41  options(District) ->
42    gen_statem:call(District, options).
43
44  -spec enter(passage(), creature()) -> ok | {error, any()}.
45  enter(District, Creature) ->
46    gen_statem:call(District, {enter, Creature}).
47
48  -spec take_action(passage(), creature_ref(), atom()) -> {ok, passage()} |
49    ↪ {error, any()}.
50  take_action(From, CRef, Action) ->
51    gen_statem:call(From, {take_action, CRef, Action}).
52
53  -spec shutdown(passage(), pid()) -> ok.
54  shutdown(District, NextPlane) ->

```

```

54     gen_statem:call(District, {shutdown, NextPlane}).
55
56 -spec trigger(passage(), trigger()) -> ok | {error, any()} | not_supported.
57 trigger(District, Trigger) ->
58     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,
65             ↪ Data)}}};
66         false -> {error, "No Description"}
67     end;
68
69 handle_event({call, From}, options, Data) ->
70     {keep_state, Data, {reply, From, {ok, maps:keys(maps:get(connections,
71         ↪ Data)}}}};
72
73 % ignore all other unhandled events
74 handle_event({call, From}, activate, Data) ->
75     {next_state, active, Data, {reply, From, ok}};
76
77 handle_event({call, From}, {run_action, CRef, Stats}, Data) ->
78     Creatures = maps:get(creatures, Data),
79     case maps:is_key(CRef, maps:get(creatures, Data)) of
80         true -> {keep_state, Data, {reply, From, {error, "Creature is already in
81             ↪ this District"}}};
82         false -> case maps:get(trigger, Data) of
83             none -> Creature1 = none, Creatures1 = none;
84             Trigger -> case run_trigger(Trigger, entering, {CRef,
85                 ↪ Stats}, Creatures) of
86                 {error, _} -> Creature1 = none, Creatures1
87                     ↪ = none;
88                 {Creature1, Creatures1} -> {Creature1,
89                     ↪ Creatures1}
90             end
91         end,
92         case {Creature1, Creatures1} of
93             {none, none} -> case maps:get(trigger, Data) of
94                 none -> NewCreatures =
95                     ↪ maps:put(CRef, Stats,
96                     ↪ maps:get(creatures, Data)),
97                     NewData =
98                     ↪ maps:update(creatures,
99                     ↪ NewCreatures, Data),
100                 {keep_state, NewData,
101                     ↪ {reply, From, ok}};

```



```

91         _ -> {keep_state, Data, {reply,
92             ↳ From, {error, "Trigger didn't
93             ↳ run"}}}}
94         end;
95         {{Ref1, Stats1}, _} ->
96             NewCreatures = maps:put(Ref1, Stats1, maps:get(creatures,
97                 ↳ Data)),
98             NewData = maps:update(creatures, NewCreatures, Data),
99             {keep_state, NewData, {reply, From, ok}}
100     end;
101     % Handle Enter on other states
102     handle_event({call, From}, {enter, _}, Data) ->
103         {keep_state, Data, {reply, From, {error, "Can't enter in this state"}}};
104     % Shutdown can be called in any state
105     handle_event({call, From}, {shutdown, NextPlane}, Data) ->
106         NextPlane ! {shutting_down, From, maps:to_list(maps:get(creatures,
107             ↳ Data))},
108         {next_state, shutting_down, Data, {next_event, internal, {From,
109             ↳ NextPlane}}};
110     handle_event({call, From}, {trigger, _Trigger}, Data) ->
111         {keep_state, Data, {reply, From, {error, "Can't set a trigger in this
112             ↳ state"}}};
113     handle_event({call, From}, {connect, _Action, _To}, Data) ->
114         {keep_state, Data, {reply, From, {error, "Can't connect in this state"}}};
115     % ignore all other unhandled events
116     handle_event(_EventType, _EventContent, Data) ->
117         {keep_state, Data}.
118
119     under_configuration({call, From}, {connect, Action, To}, Data) ->
120         case is_process_alive(To) of
121             true -> case maps:is_key(Action, maps:get(connections, Data)) of
122                 false -> Connections = maps:put(Action, To,
123                     ↳ maps:get(connections, Data)),
124                     NewData = maps:update(connections, Connections, Data),
125                     {keep_state, NewData, {reply, From, ok}};
126                 true -> {keep_state, Data, {reply, From, {error, "Action
127                     ↳ already exists"}}}
128             end;
129             false -> {keep_state, Data, {reply, From, {error, "Process not alive
130                 ↳ anymore"}}}
131         end;
132     under_configuration({call, From}, activate, Data) ->

```

```

131     {next_state, under_activation, Data, {next_event, internal, From}};
132
133
134 under_configuration({call, From}, {trigger, Trigger}, Data) ->
135     NewData = maps:update(trigger, Trigger, Data),
136     {keep_state, NewData, {reply, From, ok}};
137
138 %% General Event Handling for state under_configuration
139 under_configuration(EventType, EventContent, Data) ->
140     handle_event(EventType, EventContent, Data).
141
142 under_activation(internal, From, Data) ->
143     Result = broadcast_connection(maps:to_list(maps:get(connections, Data)),
144     ↪ From, active),
145     case Result of
146     impossible -> {next_state, under_configuration, Data, {reply, From,
147     ↪ Result}};
148     active -> {next_state, active, Data, {reply, From, Result}}
149 end;
150
151 under_activation({call, From}, activate, Data) ->
152     {keep_state, Data, {reply, From, under_activation}};
153
154 under_activation({call, From}, options, Data) ->
155     {keep_state, Data, {reply, From, {ok, maps:keys(maps:get(connections,
156     ↪ Data))}}};
157
158 %% General Event Handling for state under_activation
159 under_activation(EventType, EventContent, Data) ->
160     handle_event(EventType, EventContent, Data).
161
162 active({call, From}, {enter, {Ref, Stats}}, Data) ->
163     case maps:is_key(Ref, maps:get(creatures, Data)) of
164     true -> {keep_state, Data, {reply, From, {error, "Creature is already in
165     ↪ this District"}}};
166     false -> Creatures = maps:get(creatures, Data),
167     case maps:get(trigger, Data) of
168     none -> Creature1 = none, Creatures1 = none;
169     Trigger -> case run_trigger(Trigger, entering, {Ref, Stats},
170     ↪ Creatures) of
171     {error, _} -> Creature1 = none, Creatures1 = none;
172     {Creature1, Creatures1} -> {Creature1, Creatures1}
173     end
174     end,
175     case {Creature1, Creatures1} of
176     {none, none} ->
177     case maps:get(trigger, Data) of
178     none -> NewCreatures = maps:put(Ref, Stats, maps:get(creatures,
179     ↪ Data)),

```

```

174         NewData = maps:update(creatures, NewCreatures, Data),
175         {keep_state, NewData, {reply, From, ok}};
176     _ -> {keep_state, Data, {reply, From, {error, "Trigger didn't
177         ↳ run"}}}}
177     end;
178     {{Ref1, Stats1}, NewCreatures1} -> NewCreatures = maps:put(Ref1,
179     ↳ Stats1, maps:from_list(NewCreatures1)),
179     NewData = maps:update(creatures, NewCreatures, Data),
180     {keep_state, NewData, {reply, From, ok}}
181 end
182 end;
183
184 active({call, From}, {take_action, CRef, Action}, Data) ->
185     case maps:is_key(Action, maps:get(connections, Data)) of
186     true ->
187         case maps:is_key(CRef, maps:get(creatures, Data)) of
188         false -> {keep_state, Data, {reply, From, {error, "Creature doesn't
189         ↳ exist in this district"}}};
189         true -> case maps:get(trigger, Data) of
190             none -> Creature1 = none, Creatures1 = none;
191             Trigger ->
192                 RemoveCreature = maps:remove(CRef, maps:get(creatures,
193                 ↳ Data)),
193                 RemovedData = maps:update(creatures, RemoveCreature,
194                 ↳ Data),
194                 case run_trigger(Trigger, leaving, {CRef, maps:get(CRef,
195                 ↳ maps:get(creatures, RemovedData)) of
196                 {error, _} -> Creature1 = none, Creatures1 = none;
197                 {Creature1, Creatures1} -> {Creature1, Creatures1}
198                 end
199             end,
200             case {Creature1, Creatures1} of
201             {none, none} ->
202                 case maps:get(trigger, Data) of
203                 none -> {NewData, To} = creature_leave(CRef, Action, From,
204                 ↳ Data),
204                 case NewData of
205                 error -> {keep_state, Data, {reply, From, {error,
206                 ↳ To}}};
206                 _ -> {keep_state, NewData, {reply, From, {ok,
207                 ↳ To}}}
208                 end;
209                 _ -> {keep_state, Data, {reply, From, {error, "Trigger"}}}
210             end;
211             {{Ref, Stats}, _} -> NewCreatures = maps:put(Ref, Stats,
212             ↳ maps:get(creatures, Data)),
213             NewDataCreatures = maps:update(creatures, NewCreatures, Data),

```

```

212         {NewData, To} = creature_leave(CRef, Action, From,
213         ↪ NewDataCreatures),
214     case NewData of
215         error -> {keep_state, Data, {reply, From, {error, To}}};
216         _ -> {keep_state, NewData, {reply, From, {ok, To}}}
217     end
218 end;
219 false -> {keep_state, Data, {reply, From, {error, "Action doesn't
220 ↪ exist"}}}
221 end;
222 active({call, From}, activate, Data) ->
223     {keep_state, Data, {reply, From, active}};
224
225 %% Handle Calls to active
226 active(EventType, EventContent, Data) ->
227     handle_event(EventType, EventContent, Data).
228
229 shutting_down(internal, {From, NextPlane}, Data) ->
230     Result = broadcast_shutdown(maps:to_list(maps:get(connections, Data)),
231     ↪ From, NextPlane),
232     {stop_and_reply, normal, {reply, From, Result}};
233
234 shutting_down({call, From}, activate, Data) ->
235     {keep_state, Data, {reply, From, impossible}};
236
237 shutting_down({call, From}, options, Data) ->
238     {keep_state, Data, {reply, From, none}};
239
240 shutting_down({call, From}, shutdown, Data) ->
241     {keep_state, Data, {reply, From, ok}};
242
243 %% Handle Calls to shutting_down
244 shutting_down(EventType, EventContent, Data) ->
245     handle_event(EventType, EventContent, Data).
246
247 %% Mandatory callback functions
248 terminate(_Reason, _State, _Data) ->
249     void.
250
251 code_change(_Vsn, State, Data, _Extra) ->
252     {ok, State, Data}.
253
254 % initial State under_configuration
255 init(Desc) ->
256     %% Set the initial state + data
257     State = under_configuration, Data = #{description => Desc, connections =>
258     ↪ #{}, creatures => #{}, trigger => none},

```

```

257     {ok, State, Data}.
258
259     callback_mode() -> state_functions.
260
261     %% Synchronous Call which should wait until each response
262     broadcast_shutdown([], _, _NextPlane) -> ok;
263     broadcast_shutdown([{_Action, To} | Actions], {Pid, Ref}, NextPlane) ->
264         case is_process_alive(To) of
265             true ->
266                 case term_to_binary(To) == term_to_binary(Pid) of
267                     true -> void;
268                     false -> case term_to_binary(To) == term_to_binary(self()) of
269                         true -> void;
270                         false -> gen_statem:call(To, {shutdown, NextPlane})
271                     end
272                 end;
273             false -> void
274         end,
275     broadcast_shutdown(Actions, {Pid, Ref}, NextPlane).
276
277     %% Synchronous Call which should wait until each response
278     broadcast_connection([], _, Result) -> Result;
279     broadcast_connection([{_Action, To} | Actions], {Pid, Ref}, _) ->
280         case is_process_alive(To) of
281             false -> Result1 = impossible;
282             true -> Result1 = active,
283                 case term_to_binary(To) == term_to_binary(Pid) of
284                     false -> case term_to_binary(To) == term_to_binary(self()) of
285                         true -> void;
286                         false -> gen_statem:call(To, activate)
287                     end;
288                     true -> void
289                 end
290         end,
291     broadcast_connection(Actions, {Pid, Ref}, Result1).
292
293     creature_leave(CRef, Action, {_Pid, _}, Data) ->
294         To = maps:get(Action, maps:get(connections, Data)),
295         Stats = maps:get(CRef, maps:get(creatures, Data)),
296         case is_process_alive(To) of
297             true -> case term_to_binary(self()) == term_to_binary(To) of
298                 true -> {Data, To};
299                 false -> case gen_statem:call(To, {run_action, CRef, Stats})
300                     ↪ of
301                     ok -> NewCreatures = maps:remove(CRef,
302                     ↪ maps:get(creatures, Data)),
303                     NewData = maps:update(creatures, NewCreatures,
304                     ↪ Data),
305                     {NewData, To};

```

```

303             {error, Reason} -> {error, Reason}
304         end
305     end;
306     false -> {error, "District is shutdown"}
307 end.
308
309 run_trigger(Trigger, Event, Creature, Creatures) ->
310     Self = self(),
311     spawn(fun() -> Self ! {self(), Trigger(Event, Creature,
312         ↳ maps:to_list(Creatures))} end),
313     receive
314         {_Pid, {Creature1, Creatures1}} -> {Creature1, Creatures1}
315     after
316         2000 -> {error, "didn't run function"}
317     end.

```

B Tests Listing

B.1 Question 1.1 and Question 1.2 handin/appm/tests/BB/Main.hs

```

1  module Main where
2
3  -- Put your black-box tests in this file
4
5  import Defs
6  import Utils
7  import Parser (parseDatabase)
8  import Solver (install, normalize)
9
10 import Test.Tasty
11 import Test.Tasty.HUnit
12
13
14 tests = testGroup "Unit Tests"
15     [
16         utilities,
17         parser,
18         example
19         --predefined
20     ]
21
22 utilities = testGroup "Utilities tests"
23     [
24         -- Versions
25         testCase "Version 1 <= 1" $ V [VN 1 ""] <= V [VN 1 ""] @?= True,
26         testCase "Version 1 <= 2" $

```

```

27         V [VN 1 ""] <= V [VN 2 ""] @?= True,
28     testCase "Version 2 <= 1" $
29         V [VN 2 ""] <= V [VN 1 ""] @?= False,
30     testCase "Version 1a <= Verion1z" $
31         V [VN 1 "a"] <= V [VN 1 "z"] @?= True,
32     testCase "Version 1.1 <= 1.2" $
33         V [VN 1 "", VN 1 ""] <= V [VN 1 "", VN 2 ""] @?= True,
34     testCase "Version 1.2 <= 1.1" $
35         V [VN 1 "", VN 2 ""] <= V [VN 1 "", VN 1 ""] @?= False,
36     testCase "Version 1.1a <= 1.1b" $
37         V [VN 1 "", VN 1 "a"] <= V [VN 1 "", VN 1 "b"] @?= True,
38     testCase "Version 4.0.1 <= 04.00.001" $
39         V [VN 4 "", VN 0 "", VN 1 ""] <= V [VN 04 "", VN 00 "", VN 001
40           ↳ "" ] @?= True,
41     testCase "Version 4.0.1.3 <= 4.1.2" $
42         V [VN 4 "", VN 0 "", VN 1 "", VN 3 ""] <= V [VN 4 "", VN 1 "",
43           ↳ VN 2 ""] @?= True,
44     testCase "802.11 <= 802.11n" $ V [VN 802 "", VN 11 ""] <= V [VN 802
45       ↳ "", VN 11 "n"] @?= True,
46     testCase "802.11n <= 802.11ax" $ V [VN 802 "", VN 11 "n"] <= V [VN
47       ↳ 802 "", VN 11 "ax"] @?= True,
48     testCase "802.11ax <= 802.11bb" $ V [VN 802 "", VN 11 "ax"] <= V [VN
49       ↳ 802 "", VN 11 "bb"] @?= True,
50     -- Merge Constraints
51     testCase "Merge 2 Empty Lists" $ merge [] [] @?= Just [],
52     testCase "Merge non empty and empty List" $ merge
53       [(P "Test", (False, V [VN 0 ""] , V [VN 1 ""] ))] [] @?=
54       Just [(P "Test", (False, V [VN 0 ""] , V [VN 1 ""] ))],
55     testCase "Merge 2 non empty Lists" $ merge
56       [(P "Test", (False, V [VN 0 ""] , V [VN 1 ""] ))] [(P "Test",
57         ↳ (False, V [VN 0 ""] , V [VN 1 ""] ))] @?=
58       Just [(P "Test", (False, V [VN 0 ""] , V [VN 1 ""] ))],
59     testCase "Merge True and False" $ merge
60       [(P "Test", (True, V [VN 2 ""] , V [VN 8 ""] ))] [(P "Test",
61         ↳ (False, V [VN 4 ""] , V [VN 6 ""] ))] @?=
62       Just [(P "Test", (True, V [VN 4 ""] , V [VN 6 ""] ))],
63     testCase "Merge True and False 2nd example" $ merge
64       [(P "Test", (True, V [VN 4 ""] , V [VN 6 ""] ))] [(P "Test",
65         ↳ (False, V [VN 3 ""] , V [VN 8 ""] ))] @?=
66       Just [(P "Test", (True, V [VN 4 ""] , V [VN 6 ""] ))],
67     testCase "Merge False and False" $ merge
68       [(P "Test", (False, V [VN 4 ""] , V [VN 6 ""] ))] [(P "Test",
69         ↳ (False, V [VN 3 ""] , V [VN 8 ""] ))] @?=
70       Just [(P "Test", (False, V [VN 4 ""] , V [VN 6 ""] ))],
71     testCase "Merge False and True" $ merge
72       [(P "Test", (False, V [VN 4 ""] , V [VN 6 ""] ))] [(P "Test",
73         ↳ (True, V [VN 3 ""] , V [VN 8 ""] ))] @?=
74       Just [(P "Test", (True, V [VN 4 ""] , V [VN 6 ""] ))],
75     testCase "Merge Many Constrints example" $ merge

```

```

66         [(P "Test", (True, V [VN 4 ""] , V [VN 6 ""] )), (P "Test2",
67         ↪ (False, V [VN 3 "a"], V [VN 9 ""])),
68         (P "Test3", (False, V [VN 1 ""] , V [VN 10 ""]))]
69     [(P "Test", (False, V [VN 3 ""] , V [VN 8 ""] )), (P "Test2",
70     ↪ (False, V [VN 3 "z"], V [VN 7 ""]))] @?=
71     Just [(P "Test", (True, V [VN 4 ""] , V [VN 6 ""])), (P
72     ↪ "Test2", (False, V [VN 3 "z"], V [VN 7 ""])), (P "Test3",
73     ↪ (False, V [VN 1 ""] , V [VN 10 ""]))]
74
75     testCase "Merge same Version" $ merge
76         [(P "Test", (False, V [VN 1 ""] , V [VN 1 ""] ))] [] @?=
77         Just [(P "Test", (False, V [VN 1 ""] , V [VN 1 ""] ))]
78
79     ]
80
81     parser = testGroup "parser"
82     [
83         testCase "parse 3 packages with names" $
84             parseDatabase "package {name foo} --comment\n package {name
85             ↪ foo}package {name foo}" @?=
86             Right db1,
87         testCase "parse package with name and description" $
88             parseDatabase "package {name foo;description \"test\"}" @?=
89             Right db2,
90         testCase "parse package with name and description" $
91             parseDatabase "package {name foo;description \"test\"}" @?=
92             Right db2,
93         testCase "parse package with name, description, version" $
94             parseDatabase "package {name foo; version 1.2; description
95             ↪ \"test\"}" @?=
96             Right db3,
97         testCase "parse package with name, description, version and string"
98             ↪ $
99             parseDatabase "package {name foo; version 1.2a; description
100             ↪ \"test\"}" @?=
101             Right db4,
102         testCase "longer Version" $
103             parseDatabase "package {name foo; version 1a.2a.45;
104             ↪ description \"test\"}" @?=
105             Right db5,
106         testCase "longer Version" $
107             parseDatabase "package {name foo; version 1a.2a.45;
108             ↪ description \"test\"}" @?=
109             Right db5,
110         -- pName hyphen, end hyphen also allowed
111         testCase "Package name hypens" $
112             parseDatabase "package {name 123-wewe-RR-}" @?=
113             Left "\"Parse Error\" (line 1, column 27):\nunexpected
114             ↪ \"}\"\"nexpecting letter, digit, \"-\" or \"\\\\\"\\\\\"",
115         testCase "Package name strings" $

```



```

104         parseDatabase "package {name \"123-wewe-RR\"}" @?=
105         Right (DB [Pkg (P "123-wewe-RR") (V [VN 1 ""]) "" []]),
106     testCase "Double High comma equals 1 highcomma" $
107         parseDatabase "package {name \"123\\\"\\\"}\"" @?=
108         Right (DB [Pkg {name = P "123\\", ver = V [VN 1 ""], desc =
109             ↪ "", deps = []}],
110     testCase "Double High comma equals 1 highcomma desc" $
111         parseDatabase "package {name \"123\\\"; description \\\"\\\"\\\"}\""
112             ↪ @?=
113         Right (DB [Pkg {name = P "123\\", ver = V [VN 1 ""], desc =
114             ↪ "\\\"", deps = []}],
115     -- Case doesn't matter for keywords
116     testCase "Case insensitiveness" $
117         parseDatabase "pAckAgE {nAmE foo; vErSiOn 1a.2a.45;
118             ↪ deSCripTion \"test\\\"}" @?=
119         Right db5,
120     -- Dependencies Tests
121     testCase "Deps conflicts and requires" $
122         parseDatabase "package {name foo2; version 1a.2a.45;
123             ↪ description \"test\\\"; requires foo < 2}" @?= --requires
124             ↪ foo < 1.2 , foo >= 3;
125         Right (DB [Pkg {name = P "foo2", ver = V [VN 1 "a",VN 2
126             ↪ "a",VN 45 ""]},
127             desc = "test", deps = [(P "foo",(True,V [VN 0 ""]),V [VN 2
128             ↪ ""])])),
129     testCase "Deps requires range overwrite" $
130         parseDatabase "package {name foo2; requires foo < 3 , foo >=
131             ↪ 8.0.0}" @?=
132         Right (DB [Pkg {name = P "foo2", ver = V [VN 1 ""]}, desc =
133             ↪ "",
134             deps = [(P "foo",(True,V [VN 3 ""]),V [VN 8 "","VN 0 "","VN 0
135             ↪ ""])])),
136     testCase "Deps self referential" $
137         parseDatabase "package {name foo; requires foo < 3 , foo >=
138             ↪ 8.0.0}" @?=
139         Right (DB [Pkg {name = P "foo", ver = V [VN 1 ""]}, desc =
140             ↪ "", deps = []]),
141     testCase "Deps requires fixed range" $
142         parseDatabase "package {name foo2; requires foo < 3,
143             ↪ foo >= 8.0.0a}" @?=
144         Right (DB [Pkg {name = P "foo2", ver = V [VN 1 ""]},
145             ↪ desc = "",
146             deps = [(P "foo",(True,V [VN 3 ""]),V [VN 8 "","VN 0
147             ↪ "","VN 0 "a"])])),
148     testCase "Deps requires fixed range requires and conflicts" $
149         parseDatabase "package {name foo2; requires foo < 3
150             ↪ , foo >= 8.0.0a; conflicts bar < 3 , bar >= 8}"
151             ↪ @?=

```

```

134         Right (DB [Pkg {name = P "foo2", ver = V [VN 1 ""],
135             ↪ desc = "",
136             ↪ deps = [(P "foo", (True, V [VN 3 ""], V [VN 8 "", VN 0
137             ↪ "", VN 0 "a"])),
138             ↪ (P "bar", (False, V [VN 3 ""], V [VN 8 ""]))]]]),
139 testCase "Deps different package names" $
140     parseDatabase "package {name foo2; requires foo <
141     ↪ 3, bar >= 8.0.0a; conflicts bar < 3, foo >=
142     ↪ 8}" @?=
143     Right (DB [Pkg {name = P "foo2", ver = V [VN 1 ""],
144     ↪ desc = "",
145     ↪ deps = [(P "foo", (True, V [VN 3 ""], V [VN 8 ""])),
146     ↪ (P "bar", (True, V [VN 3 ""], V [VN 8 "", VN 0 "", VN 0
147     ↪ "a"]))] ]]),
148     -- doesn't work to change the lower, greater equal
149 testCase "Low/High changed" $
150     parseDatabase "package {name foo2; requires foo >=3
151     ↪ , bar < 8.0.0;} " @?=
152     Left "\"Parse Error\" (line 1, column
153     ↪ 38):\nunexpected \",\"\\nexpecting space,
154     ↪ \"--\", white space or \"}\"\"",
155     -- Whitespace and other more special things
156 testCase "whitespaces pkg and name" $
157     parseDatabase "package {name foo2; requires
158     ↪ foo < 3, foo >= 8.0.0a; conflicts bar
159     ↪ < 3, bar >= 8 }" @?=
160     Right (DB [Pkg {name = P "foo2", ver = V [VN 1 ""], desc = "",
161     ↪ deps = [(P "foo", (True, V [VN 3 ""], V
162     ↪ [VN 8 "", VN 0 "", VN 0 "a"])),
163     ↪ (P "bar", (False, V [VN 3 ""], V [VN 8
164     ↪ ""]))] ]]),
165     -- Comment parsing
166 testCase "Comment parsing" $
167     parseDatabase " --comment\npackage {name --comment\n foo;
168     ↪ --comment\ndescription \"test\" --comment\n}" @?=
169     Right (DB [Pkg {name = P "foo", ver = V [VN 1 ""], desc = "test",
170     ↪ deps = []}]),
171     testCase "Comment name" $
172     parseDatabase "package {name \"fo--o\"; description \"te--st\"}" @?=
173     Right (DB [Pkg {name = P "fo--o", ver = V [VN 1 ""], desc =
174     ↪ "te--st", deps = []}]),
175     -- failing to parse, comment after package
176 testCase "Comment after package" $
177     parseDatabase "package {name \"foo\"; description \"test\"}
178     ↪ --comment" @?=
179     Left "\"Parse Error\" (line 1, column 51):\nunexpected end of
180     ↪ input\nexpecting lf new-line, end of input, white space or
181     ↪ \"package\"",
182     -- Wrong Version Number

```

```

164     testCase "parse too large Version" $
165     parseDatabase "package {name \"foo\"; version 1000000}" @?=
166     Left "\"Parse Error\" (line 1, column 22):\nunexpected
      ↪ \"v\"\\nexpecting space, \"--\", white space or \"}\\\"",
167     testCase "Edge parsable" $
168     parseDatabase "package {name \"foo\"; version 999999aaaa}" @?=
169     Right (DB [Pkg {name = P "foo", ver = V [VN 999999 "aaaa"], desc =
      ↪ "", deps = []}]),
170     -- Z gets ignored, since not lowercase
171     testCase "Edge parsable" $
172     parseDatabase "package {name \"foo\"; version 999999Z}" @?=
173     Right (DB [Pkg {name = P "foo", ver = V [VN 999999 ""], desc = "",
      ↪ deps = []}]),
174     testCase "Version String too long" $
175     parseDatabase "package {name \"foo\"; version 9999aaaaaa}" @?=
176     Left "\"Parse Error\" (line 1, column 22):\nunexpected
      ↪ \"v\"\\nexpecting space, \"--\", white space or \"}\\\""
177 ]
178 where
179     ver = V [VN 1 ""]
180     pname = P "foo"
181     pname2 = P "foo2"
182     pkg = Pkg pname ver "" []
183     db1 = DB [pkg,pkg,pkg]
184     pkg2 = Pkg pname ver "test" []
185     db2 = DB [pkg2]
186     ver2 = V [VN 1 "", VN 2 ""]
187     pkg3 = Pkg pname ver2 "test" []
188     db3 = DB [pkg3]
189     ver3 = V [VN 1 "", VN 2 "a"]
190     pkg4 = Pkg pname ver3 "test" []
191     db4 = DB [pkg4]
192     ver4 = V [VN 1 "a", VN 2 "a", VN 45 ""]
193     pkg5 = Pkg pname ver4 "test" []
194     db5 = DB [pkg5]
195
196 -- Parser Example
197 example = testGroup "Example DB" [
198     testCase "Parse Example DB" $ parseDatabase "package { name foo; version
      ↪ 2.3; description \"The foo application\"; requires bar >= 1.0}
      ↪ package { name bar; version 1.0; description \"The bar library\"}
      ↪ package { name bar; version 2.1; description \"The bar library, new
      ↪ API\"; conflicts baz < 3.4, baz >= 5.0.3} package { name baz;
      ↪ version 6.1.2;}"
199     @?= Right (DB [Pkg {name = P "foo", ver = V [VN 2 "",VN 3 ""],
200         desc = "The foo application",
201         deps = [(P "bar",(True,V [VN 1 "",VN 0 ""],V [VN 1000000
      ↪ ""])))],
202         Pkg {name = P "bar", ver = V [VN 1 "",VN 0 ""],

```

```

203         desc = "The bar library", deps = []},
204     Pkg {name = P "bar", ver = V [VN 2 "", VN 1 ""],
205         desc = "The bar library, new API",
206         deps = [(P "baz", (False, V [VN 3 "", VN 4 ""]), V [VN 5 "", VN 0
207             ↪ "", VN 3 ""])]},
208     Pkg {name = P "baz", ver = V [VN 6 "", VN 1 "", VN 2 ""], desc = "",
209         ↪ deps = []}]
210 ]
211 -- just a sample; feel free to replace with your own structure
212 predefined = testGroup "predefined"
213 [testGroup "Parser tests"
214     [testCase "tiny" $
215         parseDatabase "package {name foo}package {name foo}package {name
216             ↪ foo}" @?= Right db],
217     testGroup "Solver tests"
218     [testCase "tiny" $
219         install db pname @?= Just [(pname, ver)] ] ]
220 where
221     pname = P "foo"
222     ver = V [VN 1 ""]
223     db = DB [Pkg pname ver "" []]
224
225 main = defaultMain tests

```

B.2 Question 2.1

```

1  -module(district_tests).
2  -author("silvan").
3  -include_lib("eunit/include/eunit.hrl").
4
5  district_create_test() ->
6      ?assertMatch({ok, _}, district:create("Panem")).
7
8  district_get_description_test() ->
9      {ok, P} = district:create("Panem"),
10      ?assertEqual({ok, "Panem"}, district:get_description(P)),
11      district:activate(P),
12      ?assertEqual({ok, "Panem"}, district:get_description(P)).
13
14  district_connect_districts_test() ->
15      {A, B, C} = create_districts(),
16
17      ?assertEqual(ok, district:connect(A, b, B)),
18      district:connect(A, c, C),
19      % Action c already exists in A
20      ?assertEqual(active, district:activate(A)),

```

```

21     ?assertMatch({error, _}, district:connect(A, c, C)).
22
23 district_connect2_districts_test() ->
24     {A, B, C} = create_districts(),
25
26     ?assertEqual(ok, district:connect(A, b, B)),
27     district:shutdown(C, self()),
28     %trying to connect to a terminated district
29     ?assertMatch({error, _}, district:connect(A, c, C)).
30
31 district_active_test() ->
32     {A, B, C} = create_districts(),
33
34     district:connect(A, c, C),
35     district:shutdown(C, self()),
36     % Process C not alive anymore, so A can't be activated
37     ?assertEqual(false, is_process_alive(C)),
38     ?assertEqual(impossible, district:activate(A)),
39     % B doesn't have any neighbors, so easily to be activated
40     ?assertEqual(active, district:activate(B)).
41
42 district_active2_test() ->
43     {A, _, C} = create_districts(),
44
45     district:connect(A, c, C),
46     % Activate C already, activate A later
47     ?assertEqual(active, district:activate(C)),
48     ?assertEqual(active, district:activate(A)).
49
50 district_options_test() ->
51     {A, B, C} = create_districts(),
52
53     district:connect(A, b, B),
54     district:connect(A, c, C),
55
56     ?assertEqual({ok, [b, c]}, district:options(A)),
57     ?assertEqual({ok, []}, district:options(B)),
58     ?assertEqual({ok, []}, district:options(C)).
59
60 district_enter_test() ->
61     {A, B, C} = create_districts(),
62
63     district:connect(A, b, B),
64     district:connect(A, c, C),
65
66     Bob = {make_ref(), #{}},
67     % only can enter if district active
68     ?assertMatch({error, _}, district:enter(A, Bob)),
69     district:activate(A),

```

```

70     ?assertEqual(ok, district:enter(A, Bob)).
71
72 dsitrictr_take_action_test() ->
73     {A, B, C} = create_districts(),
74
75     district:connect(A, b, B),
76     district:connect(A, c, C),
77
78     {KatnissRef, _} = Katniss = {make_ref(), #{}},
79     {PeetaRef, _} = {make_ref(), #{}},
80     district:activate(A),
81     ?assertEqual(ok, district:enter(A, Katniss)),
82     %Action doesn't exist
83     ?assertMatch({error, _}, district:take_action(A, KatnissRef, d)),
84     % Katniss stays in A
85     ?assertMatch({error, _}, district:enter(A, Katniss)),
86     %Creature hasn't joined A District
87     ?assertMatch({error, _}, district:take_action(A, PeetaRef, b)),
88     ?assertMatch({ok, _}, district:take_action(A, KatnissRef, b)),
89     % Katniss now not in District A anymore
90     ?assertEqual(ok, district:enter(A, Katniss)),
91     % But now in district B
92     ?assertMatch({error, _}, district:enter(B, Katniss)),
93     %try to move Katniss by action again to district begin
94     ?assertMatch({error, _}, district:take_action(A, KatnissRef, b)),
95     district:shutdown(B, self()),
96     ?assertMatch({error, _}, district:take_action(A, KatnissRef, b)),
97     %therefore Katniss is still in A
98     ?assertMatch({error, _}, district:enter(A, Katniss)).
99
100 district_shutdown_test() ->
101     {A, B, C} = create_districts(),
102
103     % Process is available
104     ?assertEqual(true, is_process_alive(A)),
105     ?assertEqual(true, is_process_alive(B)),
106     ?assertEqual(true, is_process_alive(C)),
107     district:connect(A, b, B),
108     district:connect(A, c, C),
109
110     ?assertEqual(ok, district:shutdown(A, self())),
111     % after shutdown undefined
112     ?assertEqual(false, is_process_alive(A)),
113     ?assertEqual(false, is_process_alive(B)),
114     ?assertEqual(false, is_process_alive(C)).
115
116 district_shutdown2_test() ->
117     {A, B, C} = create_districts(),
118

```

```

119 % Process is available
120 ?assertEqual(true, is_process_alive(A)),
121 ?assertEqual(true, is_process_alive(B)),
122 ?assertEqual(true, is_process_alive(C)),
123 district:connect(A, b, B),
124 district:connect(A, c, C),
125
126 ?assertEqual(ok, district:shutdown(B, self())),
127 % after shutdown undefined
128 ?assertEqual(true, is_process_alive(A)),
129 ?assertEqual(false, is_process_alive(B)),
130 ?assertEqual(true, is_process_alive(C)),
131 %since B already shutdown, no need to send it a shutdown message anymore
132 ?assertEqual(ok, district:shutdown(A, self())),
133 %every district should be shutdown now
134 ?assertEqual(false, is_process_alive(A)),
135 ?assertEqual(false, is_process_alive(B)),
136 ?assertEqual(false, is_process_alive(C)).
137
138 district_shutdown_cycle_test() ->
139 {A, B, _} = create_districts(),
140
141 district:connect(A, b, B),
142 district:connect(B, a, A),
143 district:connect(A,a,A),
144 %times out since cycle exists
145 district:shutdown(A,self()),
146 ?assertEqual(false, is_process_alive(A)),
147 ?assertEqual(false, is_process_alive(B)).
148
149 district_shutdown_cycle1_test() ->
150 {A, B, C} = create_districts(),
151
152 district:connect(A, b, B),
153 district:connect(B, c, C),
154 % fails if active
155 %district:connect(C, a, A),
156 %times out since cycle exists
157 district:shutdown(A,self()),
158 ?assertEqual(false, is_process_alive(A)),
159 ?assertEqual(false, is_process_alive(B)).
160
161 district_active_cycle_test() ->
162 {A, B, C} = create_districts(),
163
164 district:connect(A, b, B),
165 district:connect(B, a, A),
166 district:connect(B, c, C),
167 district:connect(C, c, C),

```

```

168   district:activate(A),
169   {Ref, _} = Katniss = {make_ref(), #{}},
170   % all connected districts get active
171   ?assertMatch(ok, district:enter(C, Katniss)),
172   district:take_action(C,Ref,c).
173
174 increment_grade(_, {CreatureRef, Stats}, Creatures) ->
175   #{grade := CurGrade} = Stats,
176   NewGrade = CurGrade + 3,
177   case NewGrade of
178     12 -> get_grade(CreatureRef, Stats, 12, happy, Creatures);
179     7 -> get_grade(CreatureRef, Stats, 7, okay, Creatures);
180     2 -> get_grade(CreatureRef, Stats, 2, okay, Creatures);
181     Grade -> get_grade(CreatureRef, Stats, Grade, sad, Creatures)
182   end.
183
184 get_grade(Ref, Stats, Grade, Mood, Creatures) ->
185   {{Ref, Stats#{grade := Grade,mood:= Mood}}, Creatures}.
186
187 district_trigger_test() ->
188   {A, B, C} = create_districts(),
189
190   district:connect(A, b, B),
191   district:connect(A, c, C),
192   district:connect(C, a, A),
193   district:connect(B, a, A),
194
195   district:trigger(A, fun increment_grade/3),
196   district:activate(A),
197   {Ref, _Stats} = Silvan = {make_ref(), #{grade => 0, mood => sad}},
198   district:enter(A, Silvan),
199   ?assertMatch({ok, _}, district:take_action(A, Ref, b)),
200   ?assertMatch({ok, _}, district:take_action(B, Ref, a)),
201   ?assertMatch({ok, _}, district:take_action(A, Ref, b)),
202   ?assertMatch({ok, _}, district:get_description(B)),
203   %Moved Silvan 4 times(entering/leaving) between A and B
204   ?assertMatch({error, _}, district:enter(B,Silvan)).
205
206 district_trigger1_test() ->
207   {A, B, C} = create_districts(),
208
209   district:connect(A, b, B),
210   district:connect(A, c, C),
211   district:connect(C, a, A),
212   district:connect(B, a, A),
213
214   % atom function
215   district:trigger(A, abc),
216   district:activate(A),

```



```

217     Silvan = {make_ref(), #{grade => 0, mood => sad}},
218     ?assertMatch({error, _}, district:enter(A, Silvan)).
219
220 cheers(_, Creature, Creatures) ->
221     timer:sleep(3000),
222     {Creature, Creatures}.
223
224 district_trigger2_test() ->
225     {A, B, C} = create_districts(),
226
227     district:connect(A, b, B),
228     district:connect(A, c, C),
229     district:connect(C, a, A),
230     district:connect(B, a, A),
231
232     % atom function
233     district:trigger(A, fun cheers/3),
234     district:activate(A),
235     Silvan = {make_ref(), #{grade => 0, mood => sad}},
236     ?assertMatch({error, _}, district:enter(A, Silvan)).
237
238 district_trigger3_test() ->
239     {A, B, C} = create_districts(),
240
241     district:connect(A, b, B),
242     district:connect(A, a, A),
243     district:connect(A, c, C),
244     district:connect(C, a, A),
245     district:connect(B, a, A),
246     {Ref, _} = Silvan = {make_ref(), #{grade => 0, mood => sad}},
247     % atom function
248
249     ?assertMatch(ok, district:trigger(A, fun cheers/3)),
250     district:activate(A),
251     ?assertMatch(ok, district:enter(B, Silvan)),
252     ?assertEqual({error, "Trigger didn't run"},
253         ↪ district:take_action(B, Ref, a)),
254     ?assertEqual({error, "Creature doesn't exist in this district"},
255         ↪ district:take_action(A, Ref, b)).
256
257 create_districts() ->
258     {ok, A} = district:create("A"),
259     {ok, B} = district:create("B"),
260     {ok, C} = district:create("C"),
261     {A, B, C}.

```

B.3 Question 2.2

```
1  -module(district_qc).
2
3  -export([territory/0, setup_territory/1]).
4  -export([prop_activate/0, prop_take_action/0]).
5
6  -include_lib("eqc/include/eqc.hrl").
7
8  % use atoms with chars from a to z
9  atom() ->
10     ?LET(S, list(eqc_gen:choose(97, 122)), list_to_atom(S)).
11
12  territory() ->
13     eqc_gen:map(eqc_gen:int(), list({atom(), eqc_gen:int()})).
14
15  create_districts([], Result) -> lists:flatten(Result);
16  create_districts([{Key, Connections} | Districts], Result) ->
17     {ok, Pid1} = district:create(Key),
18     Connect = create_connections(Pid1, Connections, []),
19     NewResult = lists:append([Pid1, Connect], Result),
20     create_districts(Districts, NewResult).
21
22  create_connections(_Pid, [], Result) -> Result;
23  create_connections(Pid, [{Atom, To} | Connections], Result) ->
24     {ok, Pid2} = district:create(To),
25     district:connect(Pid, Atom, Pid2),
26     NewResult = lists:append([Pid2], Result),
27     create_connections(Pid, Connections, NewResult).
28
29  %% Example #-4 => [{ejbdi,-1},{jennby,16}], 6 =>
30     ↪ [{fa,-12},{ta,-17},{keyj,-15}], 8 => [{w,-17}]
31  %% create all district in a map and conec
32  setup_territory(Map) ->
33     create_districts(maps:to_list(Map), []).
34
35  prop_activate() ->
36     false.
37
38  prop_take_action() ->
39     false.
```
