

AP Exam 2013

Arni Asgeirsson lwf986

November 7, 2013

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Contents

1	Introduction	4
2	Question 1: The SALSA Language	4
2.1	Fixing the grammar	4
2.1.1	Precedence and Associativity	4
2.1.2	Left factorization	5
2.1.3	Left recursion	5
2.2	My assumptions on the grammar	6
2.3	The Code	6
2.3.1	Interface functions	6
2.3.2	DataTypes	6
2.3.3	Parser functions	6
2.3.4	Helper functions	8
3	Question 2: Interpreting SALSA	8
3.1	My assumptions	8
3.2	The Code	9
3.2.1	Interface functions	9
3.2.2	Context	10
3.2.3	The Monads	10
3.2.4	Interpret functions	11
3.2.5	Interpret helper functions	12
3.2.6	Helper functions	13
4	Question 3: Atomic Transaction Server in Erlang	13
4.1	My assumptions	14
4.2	General structure of the server	14
4.2.1	Who talks to who?	14
4.2.2	The state data	14
4.2.3	The status of each transaction and the management	14
4.2.4	The pool	15
4.3	The Code	16
4.3.1	The Server API / Interface functions	16
4.3.2	Extended API	17
4.3.3	Callback functions	19
4.3.4	Helper functions	21
5	Appendix A Tests	22
5.1	Question 1 Testing	22
5.1.1	<i>Test_Parser.hs</i>	22
5.2	Question 2 Testing	23
5.2.1	<i>Test_Interp.hs</i>	24
5.3	Question 3 Testing	24
5.3.1	<i>test_at_server.erl</i>	24

6	Appendix B Grammar	29
6.1	G0	29
6.2	G1	29
6.3	G2	30
6.4	G3	31
7	Appendix C Grammar assumptions	32
7.1	My assumptions on the grammar	32
7.1.1	Case sensitivity	32
7.1.2	integer	32
7.1.3	VIdent	32
7.1.4	SIdent	32
7.1.5	White spaces	32
7.1.6	Reserved Words	32
7.1.7	Color Names	32
8	Appendix D	33
8.1	Context functions	33
8.2	Animation functions	33
9	Appendix E Code	35
9.1	at_server.erl	35
9.2	at_test.erl	40
9.3	test_at_server.erl	42
9.4	Gpx.hs	53
9.5	SalsaAst.hs	53
9.6	SalsaInterp.hs	53
9.7	SalsaParser.hs	60
9.8	Test_Interp.hs	66
9.9	Test_Parser.hs	75
9.10	multi.salsa	84
9.11	simple.salsa	84
9.12	empty.salsa	85
9.13	invalid.salsa	85

1 Introduction

The following is a report documenting my implementations of a Salsa parser, Sala interpreter written in Haskell and a multi process atomic transaction server written in Erlang, and all relevant thoughts, concerns and a discussion of interesting problems that arose while completing the work.

I believe that the greatest focus should be on these thoughts, concerns and discussions as they play a big part in the general assessment of arguments for my code, therefore these will also have great focus in this report. A note to the reader is that I tend to have many thoughts, concerns and discussions with my self, and my have a tendency to write more than others, although it is *usually* of high quality. I have noticed the very high page number, although don't fret as much of it is section headers and stuff like that stealing pages.

I am handing in a total of 13 files and these are: *src/at_server/at_server.erl*, *src/at_server/at_ex-tapi.erl*, *src/test_at_server.erl*, *src/salsa/SalsaParser.hs*, *src/salsa/SalsaInterp.hs*, *src/salsa/Test_Parser.hs*, *src/salsa/Test_Interp.hs*, *src/salsa/Gpx.hs*, *src/salsa/SalsaAst.hs*, *src/salsa/test_files/multi.salsa*, *src/salsa/test_files/simple.salsa*, *src/salsa/test_files/empty.salsa*, *src/salsa/test_files/invalid.salsa*.

Note that I have not altered the *Gpx.hs* file nor the *SalsaAst.hs* file, these are merely added for convention.

All the files are also included to this report and can be viewed in **Appendix E**.

I have decided to let all my tests be described in **Appendix A** and therefore separated from the rest of the report and individual assessments. When describing my tests I also make my final assessments on the individual parts. I use both *QuickCheck* for Haskell, *HUnit* and regular unit tests when testing.

I have used the -Wall flag when compiling my Haskell code.

Note that I am aware that the function *read* is a partial function, although I have been taught through the Advanced Programming course that this one is OK to use.

2 Question 1: The SALSA Language

The following describes my implementation of a Salsa parser in Haskell.

I have chosen to use ReadP as it allows me to check whether my grammar is ambiguous.

2.1 Fixing the grammar

The given grammar is ambiguous and this must be dealt with before proceeding with the actual implementation. The given grammar is shown in **Appendix B** and we shall call this G0.

2.1.1 Precedence and Associativity

It is assumed that the four operators $+$, $-$, $@$ and $||$ all are left-associative.

I assume that $+$ and $-$ have the same precedence, and as the assignment text specifies then $@$ has higher precedence than $||$, meaning that if we have the following pseudo input "cmd || cmd @ Vident" then this corresponds to the input "cmd || (cmd @ Vident)".

I further assume that $.$ has higher precedence than $+$ and $-$ meaning that " $r . x + c . y$ " is the same as " $(r . x) + (c . y)$ ".

The associativity is already in place and the precedence of $+$, $-$ and $.$ are also already supported by the current grammar. What is missing is that $@$ and $||$ have the correct precedence. We can ensure this by changing the grammar with the following algorithm and perform it on the non-terminal Command from G0. We might need to use the bias choice $<++$ operator instead of the fully symmetric $+++$ operator some places, but if needed we will come to that.

```

A ::= A a1 A
    | A a2 A
    | a3 .
->
A ::= A a1 B
    | B .
B ::= B a2 C
    | C .
C ::= a3 .

```

where $a1$ & $a2$ are terminals, $a3$ is something 'else' and A, B, C are non-terminals. Now $a2$ will have higher precedence than $a1$.

The resulting grammar is shown in **Appendix B** and we shall name it G1.

2.1.2 Left factorization

We can see that G1 contains the case of left factorization. We therefore fix this by doing left factorization on B.

I have summarized how I do left factorization in the following algorithm:

```

A ::= B a1
    | B a2
->
A ::= B A' .
A' ::= a1
    | a2 .

```

where A and A' are non-terminals and $B, a1$ & $a2$ could be anything.

The resulting grammar shall be called G2, and is shown in **Appendix B**.

2.1.3 Left recursion

When we look at G2 we can quickly see that there is a few cases of direct left-recursion. Therefore the next step is for me to remove left-recursion from our current grammar G2, with the following algorithm:

```

A ::= A a1
    | b1 .
->
A ::= b1 A' .
A' ::= a1 A'
    | e .

```

where A and A' are non-terminals and $a1$ & $b1$ can be anything

When applying the algorithm on G2 we get the grammar G3 which is shown in **Appendix B**.

In the rest of the report, G3 will be the referred grammar unless something else is specified.

```
data Error = NoParsePossible String
           | AmbiguousGrammar [(Program, String)]
           | UnexpectedRemainder Program String
```

Figure 1: My Error data type.

2.2 My assumptions on the grammar

All my assumptions for the grammar can be viewed Appendix D.

2.3 The Code

The code for the Salsa parser can be found in the file *SalsaParser.hs* which implements the module *SalsaParser* and exports the two functions `parseString` and `parseFile`, and the `Error` data structure. The accompanying test file is called *Test_Parser.hs* and is described in **Appendix A**.

Note that I am aware that *HLint* spits out a hint of “Reduce duplication” although I believe that fixing the hint would negatively affect the readability of my code.

2.3.1 Interface functions

`parseString :: String -> Either Error Program` This function is rather straight forward as it simply parses a given input string to a `Program` or an `Error` if the input string is invalid.

`parseFile :: FilePath -> IO (Either Error Program)` This function extends the functionality of `parseString` by reading the input string from a given file path.

2.3.2 DataTypes

Error I have defined the `Error` type to be of what is shown in Figure 1.

Here we can see that an invalid salsa program can either return that it cannot be parsed to anything at all, a parsed program and some remainder or multiply parsed programs showing that the grammar is ambiguous, or at least that the implementation of the grammar is.

2.3.3 Parser functions

These functions are generally not that interesting as they follow the grammar very strict, the following will therefore not be a very in-depth explanation, as there is not much depth to talk about.

`parse :: ReadP Program -> String -> Either Error Program` This function takes a parser `p` of type `ReadP Program` and a string `s` and parses `s` with `p` and returns the parsed program or the appropriate error if the input did not adhere to the grammar.

`runParser :: ReadP Program` This parser is a wrapper for the `pProgram` parser and simply makes sure to skip spaces both before and after parsing a `Program` and also makes sure that we are at the end of the input after parsing a program.

pProgram :: ReadP Program Parses a **Program** by trying to parse a list of definitions and or commands.

pDefComs :: ReadP [DefCom] Parses a single definition or command and then tries to parse a list of definitions and or commands.

pDefComs' :: ReadP [DefCom] Parses either a list of commands and or definitions, nothing or a symmetric choice of both.

pDefCom :: ReadP DefCom Parses either a command or a definition.

pDefinition :: ReadP Definition Parses a definition by either parsing a view definition, rectangle, circle, view or a group definition. I decided to split **pDefinition** into five individual parsers **hViewdef**, **hRectangle**, **hCircle**, **hView** and **hGroup** to make it more readable and intuitive to understand what is happening.

pCommand :: ReadP Command Is the top-level parser for parsing a command by using the **pCommand'**, **pCommand2**, **pCommand2'** and **pCommand3** parsers.

This parses a **command2** type which is either a move, bracketed or a series of '@' commands and then parses a **command'** with the previous command as incoming value

pCommand' :: Command -> ReadP Command Either parses a '|' command followed by another **command'** parsing or nothing.

pCommand2 :: ReadP Command First parses a **command3** then parses a **command2'** with the **command3** as incoming value.

pCommand2' :: Command -> ReadP Command Either parses a '@' command followed by another **command2** parse or nothing.

pCommand3 :: ReadP Command Parses either a move command or a command wrapped in a set of curly-brackets.

pVIdents :: ReadP [Ident] Parses a single vident followed by possible a list of vidents.

pVIdents' :: ReadP [Ident] Parses a list of vidents or nothing.

pSIdents :: ReadP [Ident] Parses a single sident followed by possible a list of sidents.

pSIdents' :: ReadP [Ident] Parses a list of sidents or nothing.

pPos :: ReadP Pos Parses a position of either the **Abs** or **Rel** type.

pExpr :: ReadP Expr Parses an expression by parsing a prim and then using that as incoming value to call **pExpr'**.

pExpr' :: Expr -> ReadP Expr Starts by parsing either a plus or minus expression **e** with the incoming value and then another **expr'** with **e**, otherwise it parses nothing by just returning the incoming value.

pOp :: Expr -> ReadP Expr Takes an incoming expression and either parses a plus or a minus, then the associated prim and returns either a plus or minus expression.

pPrim :: ReadP Expr Parses either an integer, an expression contained within a set of parenthesis or either an **Xproj** or **Yproj** expression.

pProj :: Ident -> ReadP Expr This parser takes an ident as an incoming value and parses either an 'x' or a 'y' and then returns the appropriate Expression of type **Xproj** or **Yproj**.

pColour :: ReadP Colour Parses one of the defined colours.

pVIdent :: ReadP Ident Simply parses a vident and skips all white spaces both before and after the word.

pSIdent :: ReadP Ident Parses a sident and, as same as **pVIdent**, also skips all white spaces both before and after the word. If the parsed word is one of the reserved words or one of the colour names then it fails, otherwise it returns the parsed word.

pInteger :: ReadP Integer Parses an integer and skips all white spaces both before and after the integer.

2.3.4 Helper functions

bracks :: Char -> ReadP b -> Char -> ReadP b Takes in two characters **a** and **b** and a parser **p**, and parses the **a** and **b** around the parsing of **p** with the use of **charT** and returns the result from **p**.

stringT :: String -> ReadP () Parses a given string **s** and skips white spaces around **s**.

charT :: Char -> ReadP () Parses a given character **c** and skips white spaces around **c**.

3 Question 2: Interpreting SALSA

In this section I will try and implement a Salsa interpreter in Haskell with the use of two monads. First I want to list the assumptions that I have taken while working on the implementation, followed by a description of my code.

3.1 My assumptions

- As the assignment text specifies then I assume that the given input **Program** is a valid **Program** in that sense that it will not cause any compile errors. Allowing me to use the Haskell function **error** to throw an error if an invalid **Program** should be used in the interpreter.

- I was unsure of what a nested **At** command should be interpreted to. I decided that an inner **At** does not consider an outer and visa versa meaning that a nested **At** command will only be visible for the innermost **At** command. Consider the case “b -> (0,0) @ A @ B”, here first B will be set as the active view, then A and then the move is performed while A is active and this no effect on B.
- I assume that when defining a group the same name cannot appear twice in the list.
- I assume that you cannot have group names inside a the list of views in a group definition.
- I assume that no two names can be used to define two things.
- I assume that the first **DefCom** will always be a view definition, as it does not make sense to perform any operations on a non-existing view.
- A shape can only be moved around within a view it has been defined, i.e. a shape that has not been defined within a view A can never appear in A.
- The list of frames goest from left to right, i.e. the first keyframe is the first element of the list and the last keyframe is the last element.
- I assume that a shape is allowed to move outside of the bounds of a view that it has been define in.

3.2 Assignment Project Exam Help

My implementation of the Salsa interpreter can be found in the file *SalsaInterp.hs* which defines and implements the module *SalsaInterp*. It exports the type **Position** and the two functions **interpolate** and **runProg**. The associated test file *SalsaInterp.hs* and is further described in **Appendix A**.

3.2.1 Interface functions

interpolate :: Integer -> Position -> Position -> [Position] This function takes an **Integer** *n*, a start **Position** *p1* and an end **Position** *p2* and interpolates between the two positions. In other words the function calculates the distribution of points in a straight line from *p1* to *p2*, including *p2* and excluding *p1*, where the number of calculated points are equal to *n*, where their is an equal distance from any point to the neighbor point.

The function assumes that *n* is non-negative.

It is implemented by considering a straight line between *p1* and *p2* and then 'slice' it up in *n* equal sizes and then returning each 'slice' point and including the end point, or excluding the start point if you will. I have avoided using floating point values, and the chaos they apparently brought with them, to consider the length of each slice to be from 0 – 100 instead of 0.0 – 1.0, which does lose me some precession, but I gain a lot more readable code and as we have to return integers anyway, the loss is very slim.

runProg :: Integer -> Program -> Animation This function initiates and runs an interpretation of a giving **Program** *p* with a given frame rate *n*, and returns the produced **Animation**. It initiates a **Salsa** monad with an empty context and then interprets each **DefCom** in the given **Program**.

3.2.2 Context

As one can see in Figure 2 I have decided to let the context be a two part thing, where the first part represents the read-only environment, which is read-only for the `SalsaCommand` monad and the second part is the read-write state.

I have named the first part `ConEnvironment` which is a type alias for a tuple of three elements: an `Environment` for binding ids to definitions, a list of the active views and an integer representing the given frame rate.

The second part is named `State` and is a mapping from an identifier `Ident` to a list of views and positions, describing the current location of each shape in each view they have been defined in.

```
data Context = Context ConEnvironment State
type ConEnvironment = (Environment, [Ident], Integer)
type Environment = M.Map Ident Definition
type State = M.Map Ident [(Ident,Position)]
```

Figure 2: My implementation of the `Context` data type.

I have also implemented a few functions to work on a `Context` and `Animation`. These are described in **Appendix D**.

3.2.3 The Monads

I have created two monads that when used together does the interpretation of a given `Program`. The first one is named `SalsaCommand` and its type declaration can be seen in Figure 3.

```
newtype SalsaCommand a = SalsaCommand {runSC :: Context -> (a,State)}
```

Figure 3: My definition of the `SalsaCommand` monad.

As one might notice then it takes a `Context` and returns some value `a` and a `State`, disallowing the monad to manipulate with the given `Context`.

The second one is the `Salsa` monad which has the declaration that can be seen in Figure 4.

```
data Salsa a = Salsa ((Context,Animation) -> (a,(Context,Animation)))
```

Figure 4: My declaration of the `Salsa` monad.

This is a state monad that carries around a `Context` and the `Animation` that is being built. I decided to let the `Salsa` monad carry these two around to easily manipulate with the two and return the completed `Animation` once done.

The actual Monad instantiating of the two types shown above can be seen in **Appendix E**.

My overall strategy for using the two monads is to let `Salsa` be the top dog and the interface to the actual interpretation, it self interprets all definitions and builds up the context environment

(part 1) and it then uses the `SalsaCommand` monad to interpret the commands. After the `Salsa` monad has let a `SalsaCommand` monad interpret a command and return the next `State` of the `Context`, the `Salsa` monad compares the previous `State` with the new returned `State` and generates all the needed draw instructions by interpolation between the two `States` for each shape in each view where it has moved.

`Salsa` then updates the global `Context` to contain the new `State` returned from the `SalsaCommand`.

The following is a short description of a set of functions to update the monads.

askCmd :: SalsaCommand Context This function simply fetches the `Context` inside a `SalsaCommand` monad.

updateState :: (Context -> State) -> SalsaCommand () `updateState` takes a function `f` that performs some operation on a `Context` and returns a `State` and performs `f` on the `Context` within `SalsaCommand` to alter the eventual returned `State`.

askCont :: Salsa Context Does the same as `askCmd`, except now we fetch the `Context` inside a `Salsa` monad instead.

runSalsa :: Context -> Salsa a -> (a,(Context,Animation)) As the name implied this function takes some initial `Context`, a `Salsa` monad and runs the monad, i.e. starts it.

updateContext :: (Context -> Context) -> Salsa () This function takes a function `f` that takes a `Context` and returns a `Context`, and applies this function `f` to the inner `Context` of a `Salsa` monad.

updateAnimation :: (Animation -> Animation) -> Salsa () `updateAnimation` takes a function `f`, that performs some operations on an `Animation` and returns another `Animation`, and applies `f` to the `Animation` part of the state of a `Salsa` monad.

3.2.4 Interpret functions

command :: Command -> SalsaCommand () The `command` function interprets a `Command` cmd within a `SalsaCommand` monad.

If `cmd` is a `Move` command then for each given shape id `sid` we must for each view that `sid` is located in, check if any of these are one of the currently active views, and if so we must update our position accordingly to the `Move` command.

If is an `At` command then we must update the active views to be of some temporary value, then execute the given command and then revert the active views to their previous self. But, as the `SalsaCommand` cannot change the `ConEnvironment` part, then we cannot update the currently active views, what we can do instead is to alter the `State` value of the `Context`.

My trick is to use the `setTmpActiveViews` helper function to go through the `State` and look at each view in each binding and if the view is one of the temporary active views then I map it to one of the actual active views, and remember this mapping. This is done through the entire `State`, followed by now mapping all the actives views that are distinct from the set of temporary views to some random value that I know no one can have, namely something with an underscore as the first letter, of course these mappings are also remembered.

Now I can run the given command with the updated `State`, and when it returns I use my mappings from before to swap back, and then I have not changed the `Context` and was still able

to update the list of active views. Although this sounds very time consuming and I bet it is, a more reasonable way would merely to update the first part of the `Context`, and then run the command, but as I cannot see how this can be done without change the type of the monad, this is not an option.

The `Par` command is a really easy one as it simply just executes the two commands.

defCom :: DefCom -> Salsa () This interpret function either interprets a `Definition` or a `Command`, if it is a definition then nothing further needs to be done beside interpreting it, but if it wants to interpret a `Command` then a `SalsaCommand` is used with the current `Context` of the running `Salsa` monad, when the monad returns then we must compare the `State` returned from the `SalsaCommand` with our own and generate a set of instructions to go from the previous state to the next. These instructions must be added to the current frame and then as we went from one keyframe to the next, we must also ensure this by beginning a new frame.

definition :: Definition -> Salsa () This function interprets a given `Definition`. If it is a view definition then we evaluate the given size expressions, add the definition to the environment, update it to be the list of active views, and add the view to the list of defined views in the `Animation`.

If the definition is a simple `View` definition meaning that it is not the same as above, but a statement to change the set of active views, then we do so by retrieving the list that corresponds to the given id and update the `Context`.

If it is a group definition then we add the binding to our environment and set the given list of view names to be the list of active views.

If we get a shape definition then we evaluate the provided values, add it to our environment, draw the shape in the current keyframe and in all current views. Furthermore we update our `State` value to contain the new shape in the current active views.

3.2.5 Interpret helper functions

evalNextPoint :: Pos -> Position -> SalsaCommand Position This function simply takes a `Pos` and a `Position` pos and evaluates it and returns either a absolute position or a relative one according to `Pos`.

setTmpActiveViews :: State -> [Ident] -> [Ident] -> SalsaCommand (State, [((Ident, Ident), Ident)])

As already mentioned then this function takes some `State`, a list of active views and a list of temporary views then it maps all occurrences of the temporary views in the `State` with one of the active views, and then follows by mapping all occurrences of the active views that are distinct from the list of temporary list with a somewhat random value prefixed with an underscore, then it returns the new state and the list of all the mappings.

revertActiveViews :: State -> [((Ident, Ident), Ident)] -> SalsaCommand State This function takes some `State s` and a list of mappings and reverts those mappings on `s` and returns this in the monad `SalsaCommand`.

setNextPosition :: Eq t => [t] -> Pos -> (t, Position) -> SalsaCommand (t, Position) This function takes a list of views `w`, a point and a tuple with a view `v` and a position. If `v` is in the list of views `w`, then the position must be updated in the tuple to be the result of the point and the position evaluated into one `Position`. If not then the same tuple that came in is also returned.

getLowestPosition :: Position -> (ViewName,Position) -> Position Returns the lowest x-coordinate and the lowest y-coordinate to be found between the two given Positions.

compareStates :: State -> State -> Salsa [GpxInstr] This function compares two States and generates draw instructions for each shape *s* and for each view *s* has been moved in.

generateInstructions :: Definition -> Integer -> [(ViewName, Position)] -> [(ViewName, Position)] -> Salsa [Frame] This function is a helper to the compareStates function and generates a list of instructions for the given Definition, frame rate, list of old positions and list of new positions.

positionToInstr :: Definition -> ViewName -> Position -> Salsa GpxInstr This functions creates a single draw instruction with for the specific shape, view and position.

evalExpr :: Monad m => Expr -> m Context -> m Integer This function takes an Expr *e* to be evaluated and something that can provide us with some context to use when evaluating Xproj and Yproj as these require that we look in the State of the Context to determine what is the lowest either *x* or *y* value of a given shape. I have made the function generic by letting the 'ask' function be a parameter so that both the Salsa and the SalsaCommand monad can use the function.

3.2.6 **Assignment Project Exam Help**

lookupKey :: Ord a => a -> M.Map a b -> b This function simply performs a lookup in a given tree with the key *a* and returns the mapped value, if no mapping was found then an error is raised.

removeDouble :: [a] -> [a] -> [a] This function takes a list *x* and a list *y* and returns the list *z*, where *z* is the list that is left of *x* when all elements that both exist in *x* and *y* has been pulled out of *x*.

evalColour :: Colour -> ColourName evalColour merely returns the respected string representation for a Colour type.

getLast :: [a] -> ([a],[a]) getLast takes in a list and returns a tuple where the first element is the whole input list expect for the last element and the second element of the tuple is the last element of the list wrapped in a list.

4 Question 3: Atomic Transaction Server in Erlang

In this section I will implement an atomic transactions server using Erlang.

I decided to use OTP for implementing my atomic transaction server, as this allows me to save a lot of time building up the server from the ground up, and provides me and the reader with a standardized interface and behavior. I chose *gen_server* over the other OTP like *gen_fsm* because our server is not a finite state machine but instead a server taking handling a lot of requests, where *gen_server* fits this task the best. This also explains all the 'comments' in the file.

4.1 My assumptions

- I always assume that the given process id, of an atomic transaction server, to all the interface function is indeed an atomic transaction server. I do no error handling nor check on this input.
- I make no assumptions on the given input State.
- I assume that no one tries to tamper with the server nor the transactions by guessing their process ids and sending them unwanted messages.

4.2 General structure of the server

Before proceeding to my actual implementation I want try and describe the overall structure of my atomic transaction server (ATS) and its managed transactions and requests, and some of the problems that arose.

4.2.1 Who talks to who?

When a new ATS has been started it is started in a new process and no transactions can be started nor do any live at startup. When a client wants to start a transaction the API function `begin_t/1` is called and this makes the ATS spawn a new transaction, which is in fact an ATS process in its self but 'loops' with another state data and expects different requests, but referred to as a transaction or 'transaction'. All communication to a transaction is done through the ATS in which they were begun. All communication is done synchronously between the ATS and its transactions, except for the `update_t/3` call and when a timeout occurs while trying to stop a transaction, then a asynchronous stop message is sent instead.

Each transaction could also just have been a tuple containing the unique reference, their current state and some status and not a whole process and the ATS would then simply just maintain a list of these, although this would disallow the opportunity to do any parallel work on the transactions which we can now with `update_t/3`, without doing some silly workaround.

4.2.2 The state data

The ATS loops with its current state, which is provided and updated by the client, and a pool of transactions. These can be idle transactions, ready to be initiated as new transactions, or ongoing transactions.

A transaction also loops with its current state and a status. This status allows the ATS to know whether the transaction is aborted, ready or idle.

4.2.3 The status of each transaction and the management

As mentioned above a transaction can have one of the three status; aborted, ready or idle. aborted means that the transaction has been aborted and any further use while in this status should return aborted. ready is when the transaction has been begun but not aborted yet. idle means that the transaction is not used for anything right now and can be initiated by the ATS to act as a fresh transaction.

Although to avoid a lot of time spent communicating and waiting for responds from non-existing processes, then the ATS maintains it self a list of each transactions, by keeping their status, process id and the associated unique reference. Where it is the unique reference that acts

as the key into this list allowing the ATS to respond quickly with `aborted` if the process does not exist, is idle or already aborted. Of course this lookup operation would always take $O(n)$ where n is the length of the list, but the alternative would be to ask the transactions each time to know what their status is, and they might be busy with some expensive update function, and the alternative would also include the same lookup operation on a list of the same length, unless the ATS does not keep track of its transactions and therefore might risk waiting for responds from a process that might never respond.

There is one time the ATS needs to ask a transaction before knowing, and this is when a transaction failed to update its state through a `update_t/3` call as this is done asynchronously the server won't know until it asks the transaction. Although once the server knows then it updates its own list, so it won't have to ask again.

There is also a slight overhead of maintaining this list but I believe that it overcomes the mentioned alternative. An improvement could be to use another data structure to that has faster lookup and replace operations, as these are the most frequent operations used.

When looking through my code one might notice that this status is not always maintained by the transaction it self, and this is because the ATS gives more credit to its own list, and will only ask transaction if it is listed to be in status ready.

To sum up the above by answering to the posed problem:

“Also, processes waiting for answers from aborted transactions must be answered with `aborted` as quickly as possible.”

My solution is to let the ATS maintain a list of the statuses of each ongoing transaction to respond quickly by avoiding the hassle to communicate with a process before answering.

4.2.4 The pool

I like to use pools. There, I said it. But I am aware that this might not be the intention in this assignment, and that is why I have defined a 'global-variable' in top of the `at_server` module `MIN_POOL` that if set to true the ATS will not keep any idle transactions in its pool, with other words, not maintain a pool.

The fact that I map the unique reference to a process id in the ATS allows me to maintain a pool of idle workers as when an idle transaction gets initiated as a new transaction, a new reference is merely created and the old association is updated to the new reference which is also returned to the client.

An argument for maintaining a pool could be that if the server is heavy used and many transactions are begun and many are aborted due to either failing functions or commits. A lot of overhead will come from destroying and creating all these processes and in such a case it could be better then let them 'stick-around' instead of being destroyed for a quicker revival. Of course if the server is barely used, a pool would be of not much use, and might even cause extra overhead to keep these processes alive. An alternative could be to add these idle processes in a sleeping-queue avoiding that they wastes any cpu cycles being idle.

As a side-note then I tried to do some small upscaled tests to test if any difference were to be found using the pool or not, but no noticeable difference were to be discovered.

4.3 The Code

The following is a description of my implementation of the *at_server* module and the *at_extapi* module, which can be found in the files *at_server.erl* and *at_extapi.erl*, and most of the interesting problems that arose. As already mentioned then I have defined the `MIN_POOL` value at the top of the *at_server* module, I have also defined a `TIME_OUT` value to allow for quick change of the accepted timeout value sent with each synchronous call to the processes, this is mostly used to go from the general default value *5000ms* to *infinity*. I advice you to not set the `TIME_OUT` value to *infinity* when running my tests as some of them will cause the server in a *dead_lock* state where the server is waiting for a transaction and visa versa.

4.3.1 The Server API / Interface functions

Note that to try and avoid letting the processes crash I have tried to catch the timeout error around each synchronous call too and within the server, although if the `TIME_OUT` value is set to something very low my `tryUpdate/0` test will fail with *timeout* in its fourth test case, which I for some reason cannot explain nor fix.

start(State) This function creates a new atomic transaction server and returns the message `{ok,Pid}` where *Pid* is the process id of the newly created ATS.

Several ATSs can be started using `start/1` and run independently.

Note that I don't make any assumptions on the given *State* value, as this should be allowed to be anything and is defined through the functions that act upon it.

stop(AT) `stop/1` makes a synchronous call to the ATS with the given process id *AT*, and orders it to shutdown and return the current state. `stop/1` will not return until all processes handled by the ATS including it self, has been shutdown, allowing the caller to be certain that when `stop/1` returns everything is cleaned up. Instead of making the call asynchronous and forcing the caller to be in doubt and ultimately the need to wait some random amount of time before the processes are shutdown, or have some unexpected behavior.

Of course with the exception that the `TIME_OUT` value has been set to something other than *infinity*.

doquery(AT, Fun) This function performs the given function *Fun* on the current state of the ATS with pid *AT* and returns the result in the form `{ok, Result}`.

If *Fun* fails *error* is returned. `doquery/2` does not update the state of the server in any way.

begin_t(AT) Here a new transaction is initialized and retrieves a copy of the current state of the given ATS, and a unique reference to the transaction is returned. Updates and queries can then be performed on the new transaction, although it belongs to the ATS that it was created within, and can therefore not be used to update the state of some other ATS.

For the unique reference, `make_ref()` is used as it guarantees that approximately the first 2^{82} created references are unique¹. One could just return the process id of the transaction, but this makes it easier for the user to send unwanted messages to the transactions, stop it or in any other way mess it up.

¹http://www.erlang.org/doc/man/erlang.html#make_ref-0

query_t(AT, Ref, Fun) `query/3` makes a synchronous call to the specified ATS which in turn makes a synchronous call to the specified transaction **Ref** and performs **Fun** on the state of **Ref** and returns the result to the original caller. The state of **Ref** is not updated in any way.

If the function fails in any way, then the transaction is aborted and **aborted** is returned to the caller.

If **Ref** is an already aborted transaction or non-existing transaction then **aborted** is returned.

update_t(AT, Ref, Fun) `update_t/3` makes an asynchronous call to the ATS and further on to the specified transaction **Ref** to update its state with the result from calling **Fun** with its current state. As the call is asynchronous `update_t/3` always returns immediately with **ok** and the caller cannot be sure whether it was a successful update or not. If the function fails then the transaction gets aborted and any future query or commit calls to that transaction **Ref** will forever return **aborted**.

If the transaction was already aborted or the given **Ref** does not exist then the update call is ignored.

commit_t(AT, Ref) To commit a transaction `commit_t/2` is used. It takes a process id **AT** of an ATS, a transaction reference **Ref** and tries to commit that transaction, i.e. update the state of the ATS with the state of **Ref**. If the given transaction has been aborted or does simply not exist, then **aborted** is returned.

If the commit was successful then the state of the ATS is updated to be the current state of the given transaction, followed by all current transactions being aborted, including the one that was committed. Meaning that with every successful commit every transaction is aborted, and all future action on any previous transaction will return aborted, or ignored if the action is `update_t/3`, and the ATS is at a 'clean' state. The assignment text specifies several times that all the *other* transactions must be aborted when doing a successful commit, but it also states that committing an already successfully committed transaction must return **aborted**, inclining that this transaction has been aborted or will be aborted no matter what, and therefore I choose that it must be aborted after it has been committed.

I have chosen to make `commit/2` a synchronous call to the ATS to let it be possible to return the proper return value to the caller, and to lock the master while he is committing to try and disallow any outside interference.

get_pids(AT) `get_pids/1` is an interface function that I have added for the sake of testing, it makes a synchronous call to a given ATS and returns a list of process id over all the living processes that the given ATS is managing, including it self.

4.3.2 Extended API

All the four functions of the extended API are blocking functions.

abort(AT, Ref) This function forces a transaction **Ref** to be aborted. I have implemented this functionality by querying the transaction with a function that always raises an error, as shown below in Figure 5.

```

abort(AT, Ref) ->
  at_server:query_t(AT,Ref,fun(_) -> error(force_abort) end).

```

Figure 5: My implementation of the Erlang function `abort/2`.

This forces the query function to fail and cause the transaction to be aborted.

By using `query_t/3` I also cause the `abort/2` function to block, i.e. by making a synchronous call to the server. I could have used `update_t/3` with the same error-raising-function, and allow `abort/2` to return immediately, this would though open up for the possibility that we could try and query, update or even commit the transaction successfully due to the nature of the message order, before our error-function is run and aborts the transaction. By using `query_t/3` I can guarantee the caller that when `abort/2` returns the transaction is indeed aborted.

I have defined the return value of `abort/2` to be the return value of `query_t/3`, which implies that `abort/2` can only return `aborted`.

tryUpdate(AT, Fun) As the name implies; `tryUpdate/2` tries to update a specified ATS with the given function `Fun`, if the `Fun` fails then error is returned but if a transaction is successfully updated with `Fun` then the return value of `tryUpdate/2` is the return value of the commit attempt.

To allow `tryUpdate/2` to return `error` if `Fun` fails I start by querying the spawned transaction and checking the result. Then to avoid performing `Fun` twice, as it may be expensive, the provided update function to the transaction is simply just a function that does not care about the previous state and merely returns the result from the query. Although this does add a slight unnecessary overhead of transporting the result from the query back and forth, but allows us to easily return `error` when `Fun` fails.

The `tryUpdate/2` function starts by beginning a transaction within AT, then does the query and if it succeeds then the transaction is updated and at last committed.

ensureUpdate(AT, Fun) `ensureUpdate/2` takes a pid of an ATS and a function `Fun` and promises to update the ATS with the function `Fun` and return `ok`, except if `Fun` fails, then `error` is returned.

I felt that the assignment text was not clear of which state `ensureUpdate/2` should update. As one can easily imagine the case of someone else making a commit before `ensureUpdate/2` is able to actual update a transaction and performing a commit, forcing the function to try again. But which state should `ensureUpdate/2` now update, the new state of the ATS or the initial state that the AST had when `ensureUpdate/2` was first called? I decided to go with the second case, as I felt that this would make most sense. Although this approach would mean that we have to rollback those commits that snuck in while we were trying to update the state of the ATS, but this also allows us to only compute the function `Fun` one time instead of recalculating it whenever we have to try again, because someone else made a commit before us, and if `Fun` is expensive; this could go on for a very long time.

`ensureUpdate/2` starts by beginning a new transaction T and then query T with `Fun`, if the result is aborted then I conclude that `Fun` failed and return `error`. If some new state S was returned start a loop function `ensureLoop/2` that will keep trying to update the state of the ATS to be S, until it succeeds. This ensures that it was the initial state of the ATS that is updated and not the in-between committed states.

choiceUpdate(AT, Fun, Val_list) `choiceUpdate/3` takes, as always, a pid for an ATS, a function `Fun(State,E)` and a list of element `Val_list`, where `State` is the state of the ATS and `E` is an element from `Val_list`. `choiceUpdate/3` then tries for each element in `Val_list` to update the state of the ATS with `Fun` in parallel. The first one to finish its update function is also the one that `choiceUpdate/3` tries to commit and the return value is then the return value from that commit. If all update functions fail then `error` is returned.

It is assumed that `Val_list` is indeed a list.

The function starts by beginning an amount of transactions equal to the number of elements in `Val_list`. Then foreach of these transactions they are asked to update their state with `Fun` and the respective element `E`. The update function has been altered to include a try and catch around `Fun`, which makes sure to send the correct message back to us. We then enter a loop function called `choiceLoop/2` where we wait for the first transaction to succeed its update function or that all transactions failed to update their state.

An important and somewhat trivial aspect of `choiceUpdate/3` is that all the transactions must have been begun before any one of them starts updating otherwise one transaction might have finished its update function before another had even been created. Therefore all are begun before any one is asked to update, to ensure a greater fairness to each transaction. Of course running through the list and asking each to update one by one, also favors the first transactions in the list, but this is how far the fairness is possible to stretch in this case.

Parallelism is achieved by utilizing that `update_t/3` is called asynchronously, and that each transaction runs in its own process.

The assignment text specifies that `choiceUpdate/3` must return the value of the commit of the first transaction to update successfully, but this is not possible if none update succeeds. Instead of letting the function hang and wait forever for some transaction to update successfully, I have added the try-catch inside the update function to be able to let `choiceUpdate/3` know if some transaction was aborted and ultimately allowing it to know if all fail and then return `error`.

There is also the slight chance that someone else makes a commit before any of our transactions gets through, `choiceUpdate/3` will still could the commit with the first successful transaction to update, if any, but as implied the return value would then be `aborted`.

As one may have noticed if glanced upon my implementation, I associate an unique reference to each call to `choiceUpdate/3` this is because in a previous call to `choiceUpdate/3` one of the transactions may be still working on its update function, and if it finishes while a future `choiceUpdate/3` call is waiting for responds from its own transactions then this old one while interfere and send 'false' messages to the `choiceUpdate/3`, by associated a unique reference to each session allows the function to discard any old obsolete messages.

4.3.3 Callback functions

The `format_status/2` function is not used, and as it is an optional callback function I have not implemented it.

Some of the callback functions are rather big and mixed with callbacks to both the ATS, transactions and both, therefore when describing some of these I will divide them into the three cases ATS, Transaction and Both equal to their separation in the actual file `at_server.erl`, to try and make it more readable.

init(Args) The `init/1` function is called by the *gen_server* when an atomic server or a transaction must be spawned. It is used to set the init state of the created server.

When an ATS is being created from the `start/1` interface function, then I have wrapped the initial given `State` in a tuple with the atom `server` as the first element to know this in the init function and set the init pool size to be an empty list.

If a new transaction is being created by the ATS then the atom `transaction` is passed along in the same fashion as `server` and the initial status is set to `ready`.

handle_call/3 `handle_call/3` is called by the *gen_server* when a synchronous request is send to the server. Here I have split the description into the three parts.

ATS If the `stop_at_server` request is send to an ATS then it propagates the stop request to all the transactions that it manages, before terminating it self with the reason `normal`, and the reply `{ok, State}` where `State` is the current state of the ATS.

When a `{doquery_t, {Ref, Fun}}` request is received then the ATS looks `Ref` up in its table of transactions, if it exist and has the status `ready` then we try to query the transaction with the given `Fun`, if it returns `error` then we remove it from our pool of transactions or mark it state to be `aborted` and reply `aborted`, if not then we return the result. If the look up failed then we also reply `aborted`.

If an ATS received a `begin_t` query then it creates a unique reference and then tries to find an idle transaction within its pool, which it then initializes to be an fresh and ready transaction to be used. If no transaction was found, then a new one is started spawned and the ATS adds it to its pool, at last the unique reference is then replied to the caller.

When `{commit_t, Ref}` is received then the ATS starts by look up in its pool if the `Ref` is present and in `ready` status, if it is found then the transaction is queried for its state `S` and if the transaction has not been aborted then the state of the ATS is updated to be `S`, and all transactions are either killed or given the idle status. `aborted` is replied if the look up failed or if the `doquery` request returned `error`.

If the `get_pid` request is received then the ATS replies with all process ids found in its pool and includes its own.

Transaction A transaction can receive an `initialize` request along with some `InitState` value, this forces the transaction to update its entire state data with the value of `InitState`. The transaction can also receive a `stop_at_trans` request which forces the transaction to stop, i.e. shutdown with the reason `normal`. If the transaction receive a `doquery` request but has the aborted status then it ignores the request and replies with `aborted`. The case where it does not have the `aborted` status is described below.

Both Both an ATS and a transaction can receive a `{doquery, Fun}` request and if so then `Fun` is computed with the current state and the result is replied back to the sender, if the function fails then we catch this and reply with `error` instead. We do not update the state of the ATS or transaction.

I have added a generic case to make sure that any unexpected request is simply just ignored instead of crashing the server.

handle_cast/2 When ever an asynchronous request is sent to the server then the `handle_cast/2` callback is called by the OTP.

ATS When an ATS received the request `{update_t, {Ref, Fun}}` it will try to look up the given `Ref` in its pool of transactions if it exists and is `ready`, meaning that it has not been aborted before and do exist, then the ATS sends an asynchronous update request to the transaction with the given function `Fun`. The state and pool of the ATS is not altered in any way.

Transaction If an aborted transaction receives an update request `{update, Fun}` then the request is just ignored. If it is in the `ready` status then it tries to perform the update and if the function `Fun` fails then its state is not updated but its status is set to `aborted`. If the function succeeds then the state is updated to be the result of the function and the status remains unchanged.

A transaction can also be asynchronously asked to stop with the `stop_at_trans` messages, and so it will.

Both Also here is a generic case added to make sure that any unexpected request is simply just ignored instead of crashing the server.

handle_info(Info, State) I do not utilize this function.

terminate(Reason, State) `terminate/2` is called by the `gen_server` right before the process is killed allowing you to do some clean up before shutdown. As already mentioned I have decided to do this in `handle_call/3`. Although if the ATS is terminated for some unexpected reason then I must try to stop any ongoing transaction, in case it is the ATS that is being terminated. If the reason is `normal` then I assume that everything is under control.

code_change(OldVsn, State, Extra) The function `code_change/3` is not used nor implemented to any external reason, some more default state as the `gen_server` behavior expects this callback function to be implemented.

4.3.4 Helper functions

ensureLoop(AT, Fun) Takes a pid of an ATS and a function `Fun`. It begins a transaction, updates it with `Fun` and tries to commit. If the commit succeeds then it returns `ok`, if not then it will try again. The function assumes that `Fun` never fails, and will only stop looping when a commit was successful.

choiceLoop(AT, AllTrans) `choiceLoop/2` takes an ATS and a list of tuples `AllTrans` where it assumes that the first element each tuple is of type `{ok, Ref}` where `Ref` is some transaction reference. If `AllTrans` is empty then `error` is returned, otherwise it will listen for the messages `{R, done}`, `{R, error}` and `E`, where the first message indicates that `R` is done with its update function and must be committed, and the return value of this commit is returned to the caller. If the error message is received then the reference is removed from `AllTrans` and the loop is called once again. If some unexpected message `E` is received then this is discarded and the loop function is called recursively.

It is assumed that the first instance of `{R, done}` is also the first to update its state, even though this cannot be completely guaranteed due to the process communication.

5 Appendix A Tests

5.1 Question 1 Testing

Testing of the parser implemented in the file *SalsaParser.hs* is done through the module `Test_Parser` implemented in the file *Test_Parser.hs*. I decided to only test my implementation through the interface of the `SalsaParser` module.

5.1.1 *Test_Parser.hs*

This test file implements the module `Test_Parser` which exports the function `runAllTests()` and `runAllTestsWith(n)`, where `n` is the number of *QuickCheck* test cases, 100 is the default value. This test file uses both *QuickCheck* and *HUnit*.

The test file is divided into three parts; a *QuickCheck* part, testing the parser with a valid input string, a *HUnit* part testing parsing of invalid input strings and that the correct precedence and associativity is maintained and a the third part testing that the parsing of files works as intended.

Part 1 The *QuickCheck* part defines a set of lists containing definitions of the different components of the Salsa grammar, not to be confused with the Salsa definitions described in the grammar. These are used when picking a random element to generate, when generating the different components. The generated test cases tend to get rather big and slow down the test I have therefore added a few more cases of `"move"` and `"const"` respectively to the lists `commands` and `exprList` to easily add a greater chance that the generated data won't grow to large.

The test checks that any² valid input string is parsed to the correct abstract syntax tree. When generator a test case then nothing is predefined, all values are random generated by *QuickCheck* `Gen` monad so everything is tested, although each value is restricted to be a valid value, and every command and `expr` is within parenthesis meaning that this test does not test for precedence or associativity of the operators.

So far the tests keeps succeeding every time I run it. Based on this I assume that it works for any valid input string, but not precedence and associativity.

Part 2 This part defines a lot of *HUnit* tests. These are divided into two categories: eleven precedence cases and forty-one error cases.

As these are unit tests I will only test a small subset of input area in the individual tests, but based on these tests I assume that it also works for the whole area.

Precedence The first five precedence test cases shows that the operators `'@'`, `'||'`, `'+'` and `'-'` are all left associative.

The next two show that `'@'` has higher precedence than `'||'`.

Test eight and nine show that `'+'` and `'-'` has the same precedence and therefore it is the order that matters, even though the end result will be the same as `+` and `-` in themselves are commutative.

The last two shows that `'.'` has higher precedence than both `'+'` and `'-'`.

²Of course not all possible input values but a random subset of the possible input each time the test is run.

Error The first test shows that an empty string cannot be parsed to anything and is therefore not a valid input string.

The next five tests that a sident cannot be used where a vident is expected.

Test case seven to eleven shows that a vident cannot be used in place of a sident.

The next four tests shows some of the invalid characters that cannot be used in a Salsa program.

The next two test cases tests that numbers cannot be used as vidents or sidents.

The following two tests shows that letters cannot be used in place of numbers.

Test case twenty to twenty-two show that not any parenthesis or brackets can be used when surrounding a command, group or point.

The next three test cases shows some of the unsupported operators that does not work for the Salsa language.

The next four tests shows that a reserved word nor a colour can be used in place of a sident.

The thirtieth test case shows that a colour must be one of the specified colour names.

Test next four test cases shows that with lacking white spaces you either get an error or an unexpected program.

Test number 35 shows that when defining a group the given list must be non-empty.

Test case number 36 to 38 show that the Salsa language is case-sensitive.

The next two test cases show that negative integers are not supported by the grammar.

The last test show that the above restriction can be bypassed by simply creating a expression that subtracts a number from a smaller number, which would produce a negative integer when executed.

All these tests return the expected result and I therefore assume that precedence and associativity works as intended and adheres to the grammar, furthermore that the parser returns expected error with a given invalid input string.

Part 3 This third part contains a set of five unit tests for testing the `parseFile` function and that it works as intended, it works as intended when parsing the content of the file with `parseString` yields the same result as parsing the file with `parseFile`.

The first test shows that an empty file is parsed as expected.

The second and the third test shows that valid Salsa files are also parsed as expected.

The fourth test shows that a invalid Salsa file is parsed as expected.

The fifth test case show that when parsing a file that does not exist then it returns the expected IO exception. Note that if the file do exist for some reason then this test will acts as a test on the content instead.

Based on these few simple tests that returned the expected values, I assume that function `parseFile` works as intended.

Conclusion Furthermore based on the tests described above and that they all return the expected results, I assume that my grammar and my implementation of the `SalsaParser` module works as intended.

5.2 Question 2 Testing

To test my implementation of the Salsa interpreter defined in the module *SalsaInterp* I have implemented a QuickCheck test to do so in the file *Test_Interp.hs* which defines and im-

plements the module *Test_Interp*. This module exports two functions `runAllTestsI()` and `runAllTestsIWith(n)`. Each run the same QuickCheck test but `runAllTestsIWith(n)` allows you to specify the number of test cases to run. The default is set to 100.

I am aware that *hlint* spits out a hint of “Reduce duplication” but as in the other files I believe it would hurt the readability to resolve it.

5.2.1 *Test_Interp.hs*

As I can assume that the interpreter should not concern much of errors and wrong input, then this will not be a focus in my test.

My test file is divided into two sections; a *QuickCheck* part and a *HUnit* part, I therefore find it adequate to divide my test description into two parts as well.

Part 1 I have defined a type `TestAnimation` which becomes an instance `QuickCheck.Arbitrary` and generates a valid Salsa Program `p` along with the expected Animation output from interpreting `p`. I do not intend to go in depth of the generator functions nor helper functions here, instead I want to describe my results.

The generated programs do not contain any `Par` commands and do not generate `At` commands that specify a group identifier. It does neither use `Xproj` or `Yproj` as expressions.

With that said then it works rather and and returns the expected results with every run.

Part 2 This part is a set of *HUnit* test cases that test the missing functionality from **Part 1**. The first set of tests tests the `Par` command. The case is very simple yet the expected result is returned.

The second case tests that the `At` command and `Group` definition works in the same program.

The third case tests that the `Xproj` and `Yproj` expressions work as intended.

The fourth set of tests show that the `interpolate` function returns the expected results.

Conclusion As my *QuickCheck* property holds for all Salsa Programs without the `Par` command at `At` commands with group identifiers I assume that that part of my Salsa interpreter works as intended. Furthermore based on the unit tests that try to cover what *QuickCheck* left behind, which are somewhat scraped to a very few number of tests due to a approaching deadline, I (almost) dare to assume that the rest of my Salsa interpreter also works as intended.

5.3 Question 3 Testing

The following is my tests for the atomic transaction server. I have created a single file *test_at_server.erl* and module *test_at_server* that tests the interface functions of the server and the extra API functions. The module exports the function `runTests/0` that runs all the tests.

I decided to let each test section focus on a single interface/API function at a time, and I will use the same distinction when describing my tests here.

I assume that no one tries to manipulate the servers nor transactions while I test them.

5.3.1 *test_at_server.erl*

testStart() This function test the functionality of `start/1` with two test cases.

The first test shows that we can start a server with some partly-random state value and that this is alive.

The second test shows that multiply servers can be started as expected.

Based on the two test cases I assume that my implementation of `start/1` works as intended.

testBegin() `testbegin/0` test the function `begin_t/1` with four test cases.

The first test shows that only one process is alive at start up.

Then the second test case tests that the number of living processes increase with the amount of transactions begun.

Test three shows that a transaction created with an ATS `A` cannot be used within another ATS.

The fourth test shows that the state of a not-updated transaction is the same as the ATS.

Based on this I assume that `begin_t/1` works as intended.

testStop() Here we test that `stop/1` returns all processes within a given ATS is stopped.

The first test tests that an ATS with no ongoing transactions is stopped after calling the stop function.

The second test shows that an ATS with multiply ongoing transactions are also stopped as expected if calling `stop/1`.

Based on these two simple test cases I assume that my implementation of `stop/1` works as intended.

testDoquery() Here we test the `doquery/2` API function. The intended functionality is that when provided with a function `F` it returns the same value as if `F` was run locally on the same state, that it does not update the state of the ATS and that if the function fails `error` is returned and the process is unchanged.

The first two test cases show that the query with a function `F` returns the same as running `F` locally.

The third test shows that the state of the ATS has not been changed.

The last test tests that if queried with some error-prone function then `error` is returned and the ATS remains unchanged.

Based on these test I assume that `doquery/2` works as intended.

testQuery_t() Here I test the `query_t/3` function with seven test cases. I expect that if queried with a function `F` then the result is the same as evaluating `F` locally, that `query_t/3` does not update the state of the transaction except if it fails, in which case would return aborted and the transaction remains aborted.

The first two test cases tests that if queried with a function `F` then the result is the same as using `F` locally.

The third case shows that the state of the queried transaction is unchanged.

The fourth case shows that if run with a function that will fail then `aborted` is returned as expected.

Test case five then tests that the transactions remains aborted if queried again.

Test six tests that if another transaction was started earlier then this remains intact.

The seventh test case tests that if a query is sent to some non-existing transaction then **aborted** is returned as expected.

All seven test cases works as expected and based on this I assume that my implementation of `query_t/3` works as intended.

testUpdate_t() With five test cases I test that my implementation of `update_t/3` works as intended.

The first test case tests that if we update some transaction with a function `F` then its state is updated to be the result of `F` and this works as expected.

The second test that if the given function fails then the transaction is aborted.

The third case test that if trying to update an already aborted transaction yields no change to the status of the transaction.

The fourth test tests that even though one transaction has been aborted then previous begun transactions are still working.

The fifth and last test case show that if calling update with a unrecognized transaction reference then no change has happened.

The above five tests return the expected results and I therefore assume that my implementation of `update_t/3` works as intended.

testCommit_t() This function tests my implementation of `commit_t/3` with six test cases.

The first test case tests that you do not need to update a transaction before commit it as expected the commit is successful and the state of the ATS remains the same.

The second test case show the transaction used before has now been aborted, and that trying to commit that transaction again returns **aborted** and it remains aborted.

Third test case tests that if a transaction is updated before being committed then the updated state becomes the new state of the ATS.

The fourth test case show that if the provided update function failed then when trying to commit the transaction **aborted** is returned.

The fifth test case tests that we can have several transactions going and if one is successfully committed then all are aborted and the state of ATS is updated to the expected one.

The last test case tests that **aborted** is returned if trying to call `commit_t/3` with a wrong transaction reference.

All six test cases return the expected values and based on this I assume that my implementation of `commit_t/3` is correct and works as intended.

testAbort() `abort/2` is tested in this function with five test cases.

The first test case tests that **aborted** is returned if calling `abort/2` on some transaction that has been started.

The second test show that **aborted** is returned if calling `abort/2` on some transaction that has been aborted previously.

The third show that several ongoing transactions can be aborted without trouble.

The fourth test case tests that no one else is affected when one is aborted.

The fifth test case tests that **aborted** is also returned if called with an unknown transactions reference.

All of the five test cases return the expected results and therefore I assume that **abort/2** work as intended.

testTryUpdate() This function tests the functionality of my implementation of **tryUpdate/2** with four test cases.

The first one shows that if no other transaction has been started then **tryUpdate/2** successfully updates the given ATS.

The second test case shows that if the function fails then error is returned and the state of the ATS remains unaltered.

The third case tests that if other transactions have been begun, then these are aborted if **tryUpdate/2** succeeds.

The last case tries to test that if someone else gets to sneak a commit in while **tryUpdate/2** is trying to update the state, then **aborted** is returned. Note that I wrote 'tries to' as this is very hard to test due to my implementation of **tryUpdate/2** as clearly a transaction who is waiting only *500ms* should be able to get his commit through before **tryUpdate/2** who waits *3500ms*. Since **tryUpdate/2** blocks the ATS by using **query_t/3** to do the computation, disallowing the previous transaction to commit after *500ms*. It seems like test once the query is done, then the other transaction is not allowed to commit as **tryUpdate/2** is allowed to commit unothered.

Even though the last test case is hard to set up, I believe that my implementation works as intended as it can either return **error** or the return value from the commit statement which can be **ok** or **aborted**, as shown earlier, and hence **tryUpdate/2** will also return **aborted** in the case that the commit returns **aborted**. With this in mind and the fact that the other test cases returned the expected result I assume that my implementation of **tryUpdate/2** works as intended.

testEnsureUpdate() As my naming convention might have revealed by now, this function tests my implementation of **ensureUpdate/2**, and does so with three test cases.

The first case tests that if no other transaction has been begun, **ensureUpdate/2** returns **ok** and the state of the ATS is updated accordingly.

The second case tests that if the given function fails then **error** is returned and the state of the ATS remains unchanged.

The third test case has the intention to show that if someone gets to commit before **ensureUpdate/2**, but after it has begun trying, then it will try again, and eventually rollback the other commit. But it doesn't work presumably of the query call in **ensureUpdate/2**.

Based on these three simple test cases I assume that **ensureUpdate/2** works as expected.

testChoiceUpdate() My last test function tests the functionality of the **choiceUpdate/3** function with six test cases.

The first case tests that no other transactions has been begun and **choiceUpdate/3** is run with a single element, then this gets through, **ok** is returned and the AST has been updated.

The second test case tests that if all choices fail with their function except one, then this one gets through, `ok` is returned and the AST is correctly updated.

The third case tests that the shorter function of two is the one who is chosen and gets to be committed to be AST. Although due to the internal workings of the OS, Erlang environment and the alike, I cannot guarantee that this case will always return the expected result, but I do dare to say that a process waiting `500ms` should only in the most strangest cases be allowed to (busy)wait all `500` before a process only waiting `1ms` gets scheduled to run and eventually make its commit.

The fourth test case tests that if someone else gets to commit before any of the choices have successfully updated their state then the return value is `aborted` and `choiceUpdate/3` does not get to update the AST.

The fifth case shows that if all choices fail then `error` is returned.

The sixth and last test case tests that if the given list is empty then `error` is returned.

All the above functions return the expected result and based on this I assume that my implementation works as intended (big surprise).

Conclusion Apart from some anal edge cases that I cannot provide the correct environment for, all my tests return the expected results and behave as intended, and I therefore dare to assume once more that my implementation the the `at_server` and `at_extapi` modules work as intended.

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

6 Appendix B Grammar

6.1 G0

```

Program ::= DefComs .
DefComs ::= DefCom
           | DefCom DefComs .
DefCom ::= Command
          | Definition .
Definition ::= 'viewdef' VIdent Expr Expr
             | 'rectangle' SIdent Expr Expr Expr Expr Colour
             | 'circle' SIdent Expr Expr Expr Expr Colour
             | 'view' VIdent
             | 'group' VIdent '[' VIdents ']' .
Command ::= SIdents '->' Pos
          | Command '@' VIdent
          | Command '||' Command
          | '{' Command '}' .
VIdents ::= VIdent
          | VIdent VIdents .
SIdents ::= SIdent
          | SIdent SIdents .
Pos ::= '(' Expr ',' Expr ')'
      | '+' '(' Expr ',' Expr ')' .
Expr ::= Prim
       | Expr '+' Prim
       | Expr '-' Prim
Prim ::= integer
       | SIdent '.' 'x'
       | SIdent '.' 'y'
       | '(' Expr ')'.
Colour ::= 'blue' | 'plum' | 'red' | 'green' | 'orange' .

```

6.2 G1

```

Program ::= DefComs .
DefComs ::= DefCom
           | DefCom DefComs .
DefCom ::= Command
          | Definition .
Definition ::= 'viewdef' VIdent Expr Expr
             | 'rectangle' SIdent Expr Expr Expr Expr Colour
             | 'circle' SIdent Expr Expr Expr Expr Colour
             | 'view' VIdent
             | 'group' VIdent '[' VIdents ']' .
Command ::= Command '||' Command2
          | Command2 .
Command2 ::= Command2 '@' VIdent
           | Command3 .
Command3 ::= SIdents '->' Pos

```

```

    | '{' Command '}' .
Vidents ::= Vident
    | Vident Vidents .
Sidents ::= Sident
    | Sident Sidents .
Pos ::= '(' Expr ',' Expr ')'
    | '+' '(' Expr ',' Expr ')' .
Expr ::= Prim
    | Expr '+' Prim
    | Expr '-' Prim .
Prim ::= integer
    | Sident '.' 'x'
    | Sident '.' 'y'
    | '(' Expr ')' .
Colour ::= 'blue' | 'plum' | 'red' | 'green' | 'orange' .

```

6.3 G2

```

Program ::= DefComs .
DefComs ::= DefCom DefComs' .
DefComs' ::= DefComs
    | e .
DefCom ::= Command
    | Definition .
Definition ::= 'viewdef' Vident Expr Expr
    | 'rectangle' Sident Expr Expr Expr Expr Colour
    | 'circle' Sident Expr Expr Expr Colour
    | 'view' Vident
    | 'group' Vident '[' Vidents ']' .
Command ::= Command1 | Command2
    | Command2 .
Command2 ::= Command2 '@' Vident
    | Command3 .
Command3 ::= Sidents '->' Pos
    | '{' Command '}' .
Vidents ::= Vident Vidents' .
Vidents' ::= Vidents
    | e .
Sidents ::= Sident Sidents' .
Sidents' ::= Sidents
    | e .
Pos ::= '(' Expr ',' Expr ')'
    | '+' '(' Expr ',' Expr ')' .
Expr ::= Prim
    | Expr Rest2 .
Op ::= '+' Prim
    | '-' Prim .
Prim ::= integer
    | '(' Expr ')'

```

```

    | SIdent '.' Rest3 .
Proj ::= 'x'
    | 'y' .
Colour ::= 'blue' | 'plum' | 'red' | 'green' | 'orange' .

```

6.4 G3

```

Program ::= DefComs .
DefComs ::= DefCom DefComs' .
DefComs' ::= DefComs
    | e .
DefCom ::= Command
    | Definition .
Definition ::= 'viewdef' VIdent Expr Expr
    | 'rectangle' SIdent Expr Expr Expr Expr Colour
    | 'circle' SIdent Expr Expr Expr Expr Colour
    | 'view' VIdent
    | 'group' VIdent '[' VIdents ']' .
Command ::= Command2 Command' .
Command' ::= '||' Command2 Command'
    | e .
Command2 ::= Command3 Command2'
Command2' ::= '@' VIdent Command2'
    | e .
Command3 ::= SIdents '->' Pos
    | '{' Command '}' .
VIdents ::= VIdent VIdents'
VIdents' ::= VIdents
    | e .
SIdents ::= SIdent SIdents'
SIdents' ::= SIdents
    | e .
Pos ::= '(' Expr ',' Expr ')'
    | '+' '(' Expr ',' Expr ')' .
Expr ::= Prim Expr' .
Expr' ::= Rest2 Expr'
    | e .
Op ::= '+' Prim
    | '-' Prim .
Prim ::= integer
    | '(' Expr ')'
    | SIdent Rest3 .
Proj ::= '.' 'x'
    | '.' 'y' .
Colour ::= 'blue' | 'plum' | 'red' | 'green' | 'orange' .

```

7 Appendix C Grammar assumptions

7.1 My assumptions on the grammar

All my assumptions for the grammar can be viewed Appendix D.

Some of the non-terminals in the grammar are not specified in the grammar, and only partly described in the assignment text. I will therefore list my assumptions and the definitions that I use in my implementation.

7.1.1 Case sensitivity

I assume that Salsa is case sensitive.

7.1.2 integer

Is a non-negative integer number and can therefore be written with the following regular expression:

$$[0 - 9]^+$$

7.1.3 VIdent

As the assignment text specifies then *VIdent* is a nonempty sequence of letters, digits and underscore, which starts with a uppercase letter and can therefore be written with the following regular expression:

$$[A - Z]^+[A - Z a - z 0 - 9 _]^*$$

7.1.4 SIdent

SIdent is the same as a *VIdent* except it cannot be one of the reserved words, described below, and has to start with a lowercase letter, which can be described as:

$$[a - z]^+[A - Z a - z 0 - 9 _]^*$$

7.1.5 White spaces

White spaces are what most would expect, spaces, any tabs and newlines. As I am using ReadP I will let the the function `skipSpaces`³ define the exact representation of white spaces.

The reserved words, color names, *VIdent*, and *SIdent* are separated by at least one white space of any kind. Symbolic tokens are separated by 0 or more white spaces and so are symbolic tokens and alpha-numeric tokens from each other.

7.1.6 Reserved Words

The reserved words are: 'viewdef', 'rectangle', 'circle', 'group' and 'view'.

7.1.7 Color Names

The names of the color are also considered to be reserved words and are the following: 'blue', 'plum', 'red', 'green' and 'orange'.

³<http://hackage.haskell.org/package/base-4.6.0.1/docs/Text-ParserCombinators-ReadP.html>

8 Appendix D

8.1 Context functions

createEmptyContext :: Integer -> Context This initializes an empty context with the given Integer `n` and returns it, where `n` is the frame rate.

lookupViews :: Ident -> Environment -> [Ident] This functions takes an Ident `k` specifying either a View or a Group and an Environment `env` and tries to lookup `k` in `env` and returns the list of views the lookup returned.

The function assumes that the given `k` is the name of either a View or a Group and that it exists in the environment.

bindCommand :: Ident -> [(Ident,Position)] -> State -> State This function takes an Ident `k` and a list of Idents and Positions `l` and maps `k` to `l` in a given State and returns the new State.

addToState :: (State -> State) -> Context -> State `addToState` takes a function `f` that takes a State and returns a new State and some Context `con` and returns the result from applying `f` to the State within `con`.

bindDefinition :: Ident -> Definition -> Environment -> Environment This functions binds an Ident to a given Definition into a given Environment and returns the updated Environment.

addToEnvironment :: (Environment -> Environment) -> Context -> Context This functions takes a function `f`, that does some computations on an Environment, a Context `con` and returns the updated Context from applying `f` to the Environment within `con`.

updateActiveViews :: [Ident] -> Context -> Context This function takes a list of Idents and updates these to be the list of active views in a given Context and returns the updated Context.

placeShapeInActiveViews :: Definition -> Context -> Context This function takes a Definition `def` and a Context and updates the State within the given Context by mapping the given shape and its position in all the currently active views. The function actually works as a wrapper around `placeShapeHelper` that does the actual job, which has just been described.

`placeShapeInActiveViews` assumes that the given `def` is a shape definition, e.i. either rectangle or a circle.

8.2 Animation functions

goToNextFrame :: Animation -> Animation This function simply starts a new keyframe by appending an empty list to the back of the list of frames.

addInstructions :: [GpxInstr] -> Animation -> Animation This function adds a given list of instructions to the current keyframe. That is by appending the instructions to the last frame found in the last of frames.

placeShapeInCurrentFrame :: Definition -> [ViewName] -> Animation -> Animation This function generates instructions to draw a given shape in each view in a given list of views and add these instructions to the current keyframe. This function assumes that the given **Definition** is either a rectangle or circle. This function is a wrapper around the helper function **placeShapeFrameHelper** that does the described work.

addViewToAnimation :: (ViewName, Integer, Integer) -> Animation -> Animation **addViewToAnimation** adds a given view to the list of view definitions in a given **Animation** and returns this updated **Animation**.

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

9 Appendix E Code

9.1 at_server.erl

```

1  %%%
2  %%% Student name: Arni Asgeirsson
3  %%% Student KU-id: lwf986
4  %%%
5
6  -module(at_server).
7
8  -behaviour(gen_server).
9
10 % Interface functions
11 -export([start/1, stop/1, begin_t/1, doquery/2, query_t/3, update_t/3, commit_t/2]).
12 % Extra interface functions
13 -export([get_pids/1]).
14 % gen_server callback functions
15 -export([init/1, handle_call/3, handle_cast/2, handle_info/2, terminate/2, code_change/3]).
16
17 %%% NOTE: I do no error_handling on these values,
18 %%% therefore set them to anything other than true/false
19 %%% and int values on your own risk
20 -define(MIN_POOL,true).
21 %%% Default timeout value is 5000 ms for call/3
22 -define(TIME_OUT, 5000).
23
24 %%%
25 %%% API
26 %%%
27
28 %%% I always assume that AT is a valid at_server process, this is never
29 %%% checked and if called with invalid value may result in unexpected error, behaviour or
30 %%% and endless waiting for a never responding process.
31
32
33 start(State) ->
34     gen_server:start(at_server, {serve, State}, []).
35
36 %%% call/2 is a synchronous call
37 stop(AT) ->
38     tryCall(gen_server:call(AT, stop_at_server, ?TIME_OUT)).
39
40 doquery(AT, Fun) ->
41     tryCall(gen_server:call(AT, {doquery, Fun}, ?TIME_OUT)).
42
43 % Returns a reference
44 begin_t(AT) ->
45     tryCall(gen_server:call(AT, begin_t, ?TIME_OUT)).
46
47 query_t(AT, Ref, Fun) ->
48     tryCall(gen_server:call(AT, {doquery_t, {Ref, Fun}}, ?TIME_OUT)).
49
50 %%% Cast is the async requests
51 update_t(AT, Ref, Fun) ->
52     gen_server:cast(AT, {update_t, {Ref, Fun}}).
53
54 commit_t(AT, Ref) ->
55     tryCall(gen_server:call(AT, {commit_t, Ref}, ?TIME_OUT)).
56
57 %%% Extra API

```

```

58 |%% Returns {ok, ListOfPids}
59 |get_pids(AT) ->
60 |    tryCall(gen_server:call(AT,get_pids,?TIME_OUT)).
61 |
62 |%%
63 |%% Callback functions
64 |%%
65 |
66 |%%
67 |%% Module: init(Args) -> Result
68 |%% -----Types:
69 |%% Args = term()
70 |%% Result = {ok,State} | {ok,State,Timeout} | {ok,State,hibernate}
71 |%%         | {stop,Reason} | ignore
72 |%% State = term()
73 |%% Timeout = int()>=0 | infinity
74 |%% Reason = term()
75 |%%
76 |
77 |
78 |%% ----- ATS -----
79 |init({server, Args}) ->
80 |    {ok,{Args,[]}};
81 |
82 |%% ----- Transaction -----
83 |init({transaction, Args}) ->
84 |    {ok,{Args,ready}}.
85 |
86 |%%
87 |%% Module: handle_call(Request, From, State) -> Result
88 |%% -----Types:
89 |%% Request = term()
90 |%% From = {pid(),Tag}
91 |%% State = term()
92 |%% Result = {reply,Reply,NewState} | {reply,Reply,NewState,Timeout}
93 |%%         | {reply,Reply,NewState,hibernate}
94 |%%         | {noreply,NewState} | {noreply,NewState,Timeout}
95 |%%         | {noreply,NewState,hibernate}
96 |%%         | {stop,Reason,Reply,NewState} | {stop,Reason,NewState}
97 |%% Reply = term()
98 |%% NewState = term()
99 |%% Timeout = int()>=0 | infinity
100 |%% Reason = term()
101 |%%
102 |
103 |%% I assume that no one will try and guess the pids of the transactions and send
104 |%% them random messages or try and manipulate with them being going past the api functions
105 |
106 |
107 |%% ----- ATS -----
108 |handle_call(stop_at_server, _, {State,Transactions}) ->
109 |    stopAllTransactions(Transactions),
110 |    {stop,normal,{ok,State},[]}; %% No reason to carry the state anymore
111 |
112 |%% ----- ATS -----
113 |handle_call({doquery_t, {Ref, Fun}}, _, {State,Transactions}) ->
114 |    {Reply,NewTransactions} =
115 |    case lists:keyfind(Ref,1,Transactions) of
116 |        {Ref,TrPid,ready} ->
117 |            try gen_server:call(TrPid, {doquery, Fun},?TIME_OUT) of
118 |                error ->
119 |                    case ?MIN_POOL of

```

```

120         false ->
121         {aborted, lists:keyreplace(Ref,1,Transactions,{Ref,TrPid,aborted})};
122         true ->
123         stopTransaction(TrPid),
124         {aborted, lists:keydelete(Ref,1,Transactions)}
125     end;
126     Result -> {Result, Transactions}
127 catch
128     _:_ -> {timeout, Transactions}
129 end;
130     ->
131     {aborted, Transactions}
132 end,
133 {reply, Reply, {State, NewTransactions}};
134 %% -----
135 %% ----- ATS -----
136 handle_call(begin_t, _, {State, Transactions}) ->
137     URef = make_ref(),
138     NewTransactions =
139     case lists:keyfind(idle,3,Transactions) of
140         false ->
141         {ok, TrPid} = gen_server:start(at_server, {transaction, State}, []),
142         [{URef,TrPid,ready}|Transactions];
143         {Ref,TrPid,idle} ->
144         %% Make sure to update its state to be of ours
145         try gen_server:call(TrPid,{initialize,{State,ready}},?TIME_OUT) of
146             ok ->
147             lists:keyreplace(Ref,1,Transactions,{URef,TrPid,ready})
148         catch
149             _:_ ->
150             {ok, TrPid} = gen_server:start(at_server, {transaction, State}, []),
151             [{URef,TrPid,ready}|Transactions]
152         end
153     end,
154     {reply,{ok,URef},{State,NewTransactions}};
155 %% -----
156 %% ----- ATS -----
157 handle_call({commit_t, Ref}, {State, Transactions}) ->
158     {Reply, NewState, NewTransactions} =
159     case lists:keyfind(Ref,1,Transactions) of
160         {Ref,TrPid,ready} ->
161         try gen_server:call(TrPid, {doquery,fun(I) -> I end},?TIME_OUT) of
162             error ->
163             {aborted,State,lists:keyreplace(Ref,1,Transactions,{Ref,TrPid,aborted})};
164             {ok, NS} ->
165             %% Abort all transactions now,
166             %% ei set their state to idle
167             %% Note that their state does not get 'cleaned up' this is done in begin_t
168             case ?MIN_POOL of
169                 false ->
170                 NT = lists:map(fun({R,P,_}) -> {R,P,idle} end, Transactions),
171                 {ok,NS,NT};
172                 true ->
173                 stopAllTransactions,
174                 {ok,NS,[]}
175             end
176         catch
177             _:_ -> {timeout,State,Transactions}
178         end;
179         ->
180         {aborted,State,Transactions}
181     end,

```

```

182     {reply, Reply, {NewState, NewTransactions}}};
183 %% -----
184 %% ----- ATS -----
185 handle_call(get_pids, _, {State, Transactions}) ->
186     AllPids = [self() | lists:flatmap(fun({_ , P, _}) -> [P] end, Transactions)],
187     {reply, {ok, AllPids}, {State, Transactions}};
188 %% -----
189 %% ----- Transaction -----
190 handle_call({initialize, InitState}, _, _) ->
191     {reply, ok, InitState};
192 %% -----
193 %% ----- Transaction -----
194 handle_call(stop_at_trans, _, {State, _}) ->
195     {stop, normal, {ok, State}, []}; %% No reason to carry the state anymore
196 %% -----
197 %% ----- Transaction -----
198 handle_call({doquery, _}, _, {State, aborted}) ->
199     {reply, error, {State, aborted}};
200 %% -----
201 %% ----- Both -----
202 handle_call({doquery, Fun}, _, {State, Satalite}) ->
203     Reply = try Fun(State) of
204         Result -> {ok, Result}
205     catch
206         _:_ -> error
207     end,
208     {reply, Reply, {State, Satalite}};
209 %% -----
210 %% ----- Both -----
211 handle_call(Msg, _, State) ->
212     {reply, {unrecognized_message, Msg}, State}.
213
214 %%-----
215 %% Module: handle_cast(Request, State) -> Result
216 %% -----Types:
217 %% Request = term()
218 %% State = term()
219 %% Result = {noreply, NewState} | {noreply, NewState, Timeout}
220 %%         | {noreply, NewState, nil, Reason}
221 %%         | {stop, Reason, NewState}
222 %% NewState = term()
223 %% Timeout = int()>=0 | infinity
224 %% Reason = term()
225 %%-----
226
227 %%-----
228 %% ----- ATS -----
229 handle_cast({update_t, {Ref, Fun}}, {State, Transactions}) ->
230     case lists:keyfind(Ref, 1, Transactions) of
231         {Ref, TrPid, ready} ->
232             gen_server:cast(TrPid, {update, Fun});
233     _ ->
234         do_nothing
235     end,
236     {noreply, {State, Transactions}};
237 %% -----
238 %% ----- Transaction -----
239 handle_cast({update, _}, {State, aborted}) ->
240     {noreply, {State, aborted}};
241 handle_cast({update, Fun}, {State, ready}) ->
242     NewState = try Fun(State) of
243         Result -> {Result, ready}

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

244         catch
245         _:_ -> {State, aborted}
246         end,
247         {noreply, NewState};
248     handle_cast(stop_at_trans, State) ->
249         {stop, normal, State};
250
251     %% ----- Both -----
252     handle_cast(_, State) ->
253         {noreply, State}.
254
255     %%% -----
256     %% Module:handle_info(Info, State) -> Result
257     %% -----Types:
258     %% Info = timeout | term()
259     %% State = term()
260     %% Result = {noreply, NewState} | {noreply, NewState, Timeout}
261     %%           | {noreply, NewState, hibernate}
262     %%           | {stop, Reason, NewState}
263     %% NewState = term()
264     %% Timeout = int()>=0 | infinity
265     %% Reason = normal | term()
266     %%% -----
267
268     %% ----- Both -----
269     %%
270     handle_info(_, State) ->
271         {noreply, State}.
272
273     %%% -----
274     %% Module:terminate(Reason, State)
275     %% -----Types:
276     %% Reason = normal | shutdown | {shutdown, term() | term()}
277     %% State = term()
278     %%% -----
279
280     %% ----- Both -----
281     %%
282     terminate(normal, _) ->
283         ok;
284
285     %% ----- ATS -----
286     terminate(Error, {State, [H|T]}) ->
287         io:format(
288             "#####Error: at_server with state: ~p~n"
289             ++"#####Terminating due to some unexpected error: ~p!~n", [State, Error]),
290
291         %% Try to shutdown each living transaction
292         stopAllTransactions([H|T]);
293
294     %% ----- Transaction -----
295     terminate(Error, State) ->
296         io:format(
297             "#####Error: transaction with state: ~p~n"
298             ++"#####Terminating due to some unexpected error: ~p!~n", [State, Error]),
299         ok.
300
301     %%% -----
302     %% Module:code_change(OldVsn, State, Extra) -> {ok, NewState} | {error, Reason}
303     %% -----Types:
304     %% OldVsn = Vsn | {down, Vsn}
305     %% Vsn = term()

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

306 %% State = NewState = term()
307 %% Extra = term()
308 %% Reason = term()
309 %%%
310
311 %% The code_change/3 callback is not used and therefore not really implemented,
312 %% although present due to the expected callback exports
313
314 %%%
315 %%% Both %%%
316 code_change(_, State, _) ->
317   {ok, State}.
318
319
320 %%%
321 %%% Helper server-functions
322 %%%
323
324 stopAllTransactions(Transactions) ->
325   lists:foreach(fun({_P,_}) -> stopTransaction(P) end, Transactions).
326
327 stopTransaction(Pid) ->
328   try gen_server:call(Pid, stop_at_trans, ?TIME_OUT)
329   catch
330     _:_ -> gen_server:cast(Pid, stop_at_trans)
331   end.
332
333 tryCallACallRef
334 try Call of
335   Result ->
336     Result
337   catch
338     timeout:_ ->
339       timeout
340     end.

```

9.2 at_extapi Erl

```

1 %%%
2 %%% Student name: Arni Asgeirsson
3 %%% Student KU-id: lwf986
4 %%%
5
6 -module(at_extapi).
7
8 -export([abort/2, tryUpdate/2, ensureUpdate/2, choiceUpdate/3]).
9
10 %%%
11 %%% Extended API
12 %%%
13
14 abort(AT, Ref) ->
15   at_server:query_t(AT, Ref, fun(_) -> error(force_abort) end).
16
17 tryUpdate(AT, Fun) ->
18   {ok, Ref} = at_server:begin_t(AT),
19   %% By querying the transaction first, we can be sure the function returns an error
20   %% to stick to the api, and if we do that then we do not need to recalculate that again
21   %% although adding some overhead of transporting the data back and forth
22   case at_server:query_t(AT, Ref, Fun) of

```



```

23 {ok, State} ->
24     %% No reason to evaluate the result again
25     ok = at_server:update_t(AT, Ref, fun(_) -> State end),
26     %% By now we either get a succesfull commit or got aborted
27     %% because someone else made a commit before us.
28     at_server:commit_t(AT, Ref);
29 aborted ->
30     error
31 end.
32
33
34 ensureUpdate(AT, Fun) ->
35     {ok, Ref} = at_server:begin_t(AT),
36     case at_server:query_t(AT, Ref, Fun) of
37     {ok, State} ->
38         %% ensureLoop begins a new transaction, making R obsolete
39         %% but it will be cleaned up with the next commit.
40         ensureLoop(AT, fun(_) -> State end);
41     aborted ->
42         %% There is a slight change that someone made a commit before
43         %% we were able to call query (not after we begun querying)
44         %% and after we begun the transaction. This is accepted.
45         error
46     end.
47
48 ensureLoop(AT, Fun) ->
49     {ok, R} = at_server:begin_t(AT),
50     ok = at_server:update_t(AT, R, Fun),
51     case at_server:commit_t(AT, R) of
52     ok ->
53         ok;
54     aborted ->
55         %% Ugh, we try again
56         ensureLoop(AT, Fun)
57     end.
58
59 %% Note that this does not 100% that the first is the one to
60 %% get through, (though it does locally) as the message queue is not gueranteed globally.
61 %% It is assumed that Val_list is indeed a list
62 choiceUpdate(AT, Fun, Val_list) ->
63     AllTrans = lists:map(fun(E) -> {at_server:begin_t(AT), E} end, Val_list),
64     Me = self(),
65     URef = make_ref(),
66     lists:foreach(fun({{ok, R}, E}) ->
67         ok = at_server:update_t(
68             AT,
69             R,
70             fun(State) ->
71
72                 try Fun(State, E) of
73                     Res ->
74                         info(Me, {URef, R, done}),
75                         Res
76                 catch
77                     _:_ ->
78                         info(Me, {URef, R, error}),
79                         %% Remember to fail so its
80                         %% state is updated properly
81                         Fun(State, E)
82                 end
83             end)
84     end,

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

85     AllTrans ),
86     choiceLoop(AT, AllTrans , URef).
87
88 %% Used by choiceUpdate
89 choiceLoop(_, [], _) ->
90     error;
91 choiceLoop(AT, AllTrans , URef) ->
92     %% Note that the messages are not guaranteed to arrive in the same order
93     %% they are sent, therefore it could be that R is not the one who finished first
94     %% but then again if R would send the commit message himself, we still are not
95     %% sure the someone won't skip in front of him.
96     %% If we let the transaction to themselves commit, we still need to let them
97     %% send us a message to protect against the case of where all functions fail.
98     receive
99     {URef, R, done} ->
100         at_server:commit_t(AT, R);
101     {URef, R, error} ->
102         RestTrans = lists:keydelete({ok, R}, 1, AllTrans),
103         choiceLoop(AT, RestTrans , URef);
104     _ ->
105         choiceLoop(AT, AllTrans , URef)
106     end.
107
108
109 %%%
110 %%% Communication primitives
111 %%%
112
113 %% asynchronous communication
114
115 info(Pid, Msg) ->
116     Pid ! Msg.

```

Assignment Project Exam Help

<https://powcoder.com>

9.3 test_at_server.erl

```

1 %%%
2 %%% Student name: Arni Asgeirsson
3 %%% Student KU-id: lwf986
4 %%%
5
6 -module(test_at_server).
7
8 -export([runTests/0]).
9
10 -define(SLEEP_TIME, 40).
11
12 %%%
13 %%% Interface
14 %%%
15
16 %% Run all tests
17 runTests() ->
18     io:format("_____~n"),
19     io:format("_____ Running tests for the at_server module _____~n"),
20     io:format("_____~n~n"),
21     io:format("_____ Running Unit tests _____~n"),
22     io:format("Testing start / 1:....."),
23     io:format("~p~n", [testStart()]),
24     io:format("Testing begin_t / 1:....."),
25     io:format("~p~n", [testBegin()]),

```

Add WeChat powcoder

```

26     io:format("Testing stop / 1:....."),
27     io:format("~p~n",[testStop()]),
28     io:format("Testing doquery / 2:....."),
29     io:format("~p~n",[testDoquery()]),
30     io:format("Testing query_t / 3:....."),
31     io:format("~p~n",[testQuery_t()]),
32     io:format("Testing update_t / 3:....."),
33     io:format("~p~n",[testUpdate_t()]),
34     io:format("Testing commit_t / 2:....."),
35     io:format("~p~n",[testCommit_t()]),
36     io:format("Testing abort / 2:....."),
37     io:format("~p~n",[testAbort()]),
38     io:format("Testing tryUpdate / 2:....."),
39     io:format("~p~n",[testTryUpdate()]),
40     io:format("Testing ensureUpdate / 2:....."),
41     io:format("~p~n",[testEnsureUpdate()]),
42     io:format("Testing choiceUpdate / 3:....."),
43     io:format("~p~n",[testChoiceUpdate()]).
44
45     %%%
46     %%% Test Functions
47     %%%
48
49     %%
50     %% ----- ATS API -----
51
52     %% Test start / 1
53     testStart(S) ->
54         %% Init test data
55         StateA = [asd,"d",233],
56
57         %% 1. Test that we can start a server with some state
58
59         {ok, Pid1} = at_server:start([]),
60         timer:sleep(?SLEEP_TIME),
61         Test1 = assertEquals(true,isProcessAlive(Pid1)),
62
63         %% 2. Test that we can start multiple servers
64
65         {ok, Pid2} = at_server:start(Pid1),
66         timer:sleep(?SLEEP_TIME),
67         Test21 = assertEquals(true,isProcessAlive(Pid2)),
68
69         {ok, Pid3} = at_server:start(StateA),
70         timer:sleep(?SLEEP_TIME),
71         Test22 = assertEquals(true,isProcessAlive(Pid3)),
72         Test2 = areTrue([Test21,Test22]),
73
74         %% Clean up
75         {ok,[]} = at_server:stop(Pid1),
76         {ok,Pid1} = at_server:stop(Pid2),
77         {ok,StateA} = at_server:stop(Pid3),
78
79         areTrue([Test1,Test2]).
80
81     %% Test begin_t / 1
82     testBegin() ->
83         %% Init test data
84         State = some_state,
85         {ok, Pid1} = at_server:start(State),
86         {ok, Pid2} = at_server:start(State),
87         timer:sleep(?SLEEP_TIME),

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

88
89 %% 1. Test that only one process exist on start up
90
91 {ok, AllPids} = at_server:get_pids(Pid1),
92 Test1 = assertEquals(1, length(AllPids)),
93
94 %% 2. Test that we can start transactions and these spawn a the correct amount
95 %% of transactions/processes
96
97 begin_transactions(Pid1, 1),
98 timer:sleep(?SLEEP_TIME),
99 {ok, AllPids1} = at_server:get_pids(Pid1),
100 Test21 = assertEquals(2, length(AllPids1)),
101
102 begin_transactions(Pid1, 5),
103 timer:sleep(?SLEEP_TIME),
104 {ok, AllPids3} = at_server:get_pids(Pid1),
105 Test22 = assertEquals(7, length(AllPids3)),
106
107 begin_transactions(Pid1, 28),
108 timer:sleep(?SLEEP_TIME),
109 {ok, AllPids4} = at_server:get_pids(Pid1),
110 Test23 = assertEquals(35, length(AllPids4)),
111
112 Test24 = assertEquals(true, areProcessesAlive(AllPids4)),
113 Test2 = areTrue([Test21, Test22, Test23, Test24]),
114
115 %% 3. Test that we cannot create a transaction in A and use it in B
116 {ok, R} = at_server:begin_t(Pid1),
117 Test3 = assertEquals(true, isAborted(Pid2, R)),
118
119 %% 4. Test that the state of a transaction is the same as the ATS
120 Test4 = assertEquals(at_server:query(Pid1, fun identity/1),
121   at_server:query_t(Pid1, R, fun identity/1)),
122
123 %% Clean up
124 {ok, State} = at_server:stop(Pid1),
125 {ok, State} = at_server:stop(Pid2),
126
127 areTrue([Test1, Test2, Test3, Test4]).
128
129 %% Test stop/1
130 testStop() ->
131   %% Init test data
132   State = some_state,
133   {ok, Pid1} = at_server:start(State),
134   {ok, Pid2} = at_server:start(State),
135   timer:sleep(?SLEEP_TIME),
136
137   %% 1. Test that an at_server dies after stop/1 has been called
138
139   Test11 = assertEquals(true, isProcessAlive(Pid1)),
140   Test12 = assertEquals({ok, State}, at_server:stop(Pid1)),
141   timer:sleep(?SLEEP_TIME),
142   Test13 = assertEquals(true, isProcessDead(Pid1)),
143   Test1 = areTrue([Test11, Test12, Test13]),
144
145   %% 2. Test that all initiated transactions are also stopped with the at_server
146
147   begin_transactions(Pid2, 4),
148   timer:sleep(?SLEEP_TIME),
149   {ok, AllPids} = at_server:get_pids(Pid2),

```

```

150     Test21 = assertEquals(true, areProcessesAlive(AllPids)),
151     {ok, State} = at_server:stop(Pid2),
152     timer:sleep(?SLEEP_TIME),
153     Test22 = assertEquals(true, areProcessesDead(AllPids)),
154     Test2 = areTrue([Test21, Test22]),
155
156     %% Clean up
157
158     areTrue([Test1, Test2]).
159
160
161 %% Test doquery/2
162 testDoquery() ->
163     %% Init test data
164     State = "I am not an A",
165     {ok, Pid1} = at_server:start(State),
166     timer:sleep(?SLEEP_TIME),
167
168     %% 1. Test doquery returns the state with an identity function
169     Test1 = assertEquals({ok, State},
170         at_server:doquery(Pid1, fun identity/1)),
171
172     %% 2. Test that it returns what is returned by the given function
173     Test21 = assertEquals({ok, mapMult2(State)},
174         at_server:doquery(Pid1, fun mapMult2/1)),
175     Test22 = assertEquals({ok, mapToA(State)},
176         at_server:doquery(Pid1, fun mapToA/1)),
177     Test2 = areTrue([Test21, Test22]),
178
179     %% 3. Show that the doquery doesn't update the state data
180     Test3 = assertEquals({ok, State},
181         at_server:doquery(Pid1, fun identity/1)),
182
183     %% 4. Test what happens if the function causes some error
184     Test41 = assertEquals(error,
185         at_server:doquery(Pid1, fun onlyEmpty/1)),
186     Test42 = assertEquals(true, isProcessAlive(Pid1)),
187     Test43 = assertEquals({ok, State},
188         at_server:doquery(Pid1, fun identity/1)),
189     Test4 = areTrue([Test41, Test42, Test43]),
190
191     %% Clean up
192     {ok, State} = at_server:stop(Pid1),
193
194     areTrue([Test1, Test2, Test3, Test4]).
195
196 %% Test query_t/3
197 testQuery_t() ->
198     %% Init test values
199     State = [1,2,3,4,5,6],
200     {ok, Pid1} = at_server:start(State),
201     {ok, R1} = at_server:begin_t(Pid1),
202     {ok, R2} = at_server:begin_t(Pid1),
203     timer:sleep(?SLEEP_TIME),
204
205     %% 1. Test that an unaltered trans state returns the initial state when used with identity
206     Test1 = assertEquals({ok, State},
207         at_server:query_t(Pid1, R1, fun identity/1)),
208
209     %% 2. Test that it returns the same as when run on the state here
210     Test21 = assertEquals({ok, mapMult2(State)},
211         at_server:query_t(Pid1, R1, fun mapMult2/1)),

```

```

212 Test22 = assertEquals({ok,mapToA(State)},
213   at_server:query_t(Pid1,R1,fun mapToA/1)),
214 Test2 = areTrue([Test21, Test22]),
215
216 %% 3. Show that query_t doesnt update its state
217 Test3 = assertEquals({ok, State},
218   at_server:query_t(Pid1,R1,fun identity/1)),
219
220 %% 4. Test what happens if the function causes some error
221 Test41 = assertEquals(aborted,
222   at_server:query_t(Pid1,R1,fun onlyEmpty/1)),
223 Test42 = assertEquals(true, isProcessAlive(Pid1)),
224 Test4 = areTrue([Test41, Test42]),
225
226 %% 5. Show that aborted is also returned when trying to query it again (even with a valid function)
227 Test5 = assertEquals(aborted,
228   at_server:query_t(Pid1,R1,fun identity/1)),
229
230 %% 6. Test that even though R1 is aborted R2 is still good
231 Test6 = assertEquals({ok,State},
232   at_server:query_t(Pid1,R2,fun identity/1)),
233
234 %% 7. Test that a wrong ref_id is considered to be an aborted transaction
235 WrongRef = make_ref(),
236 Test71 = assertEquals(aborted,
237   at_server:query_t(Pid1,WrongRef,fun identity/1)),
238 Test72 = assertEquals(aborted,
239   at_server:query_t(Pid1,false_wrong,fun identity/1)),
240 Test7 = areTrue([Test71, Test72]),
241
242 %% Clean up
243 {ok, State} = at_server:stop(Pid1),
244
245 areTrue([Test1, Test2, Test3, Test4, Test5, Test6, Test7]),
246
247 %% Test update_t/3
248 testUpdate_t() ->
249   %% Init test values
250   State = [1,2,3,4,5,6],
251   {ok,Pid1} = at_server:start(State),
252   {ok,R1} = at_server:begin_t(Pid1),
253   {ok,R2} = at_server:begin_t(Pid1),
254   ok = at_server:update_t(Pid1,R2,fun removeEven/1),
255   {ok,R3} = at_server:begin_t(Pid1),
256   timer:sleep(?SLEEP_TIME),
257
258   %% 1. Test that if we update it then it contains the new data
259   Test11 = assertEquals({ok,State},
260     at_server:query_t(Pid1,R1,fun identity/1)),
261   ok = at_server:update_t(Pid1,R1,fun removeEven/1),
262   timer:sleep(?SLEEP_TIME),
263   Test12 = assertEquals({ok,removeEven(State)},
264     at_server:query_t(Pid1,R1,fun identity/1)),
265   Test1 = areTrue([Test11, Test12]),
266
267   %% 2. Test what happens if the update function fails
268   %% I.e. Show that it is aborted
269   ok = at_server:update_t(Pid1,R1,fun onlyEmpty/1),
270   timer:sleep(?SLEEP_TIME),
271   Test2 = assertEquals(true, isAborted(Pid1,R1)),
272
273   %% 3. Test calling update on a aborted transaction

```

```

274 ok = at_server:update_t(Pid1,R1,fun removeEven/1),
275 timer:sleep(?SLEEP_TIME),
276 Test3 = assertEquals(true, isAborted(Pid1,R1)),
277
278 %% 4. Show that even though it is aborted R2 & R3 still maintain their state and are fully functional
279 Test41 = assertEquals({ok,removeEven(State)},
280   at_server:query_t(Pid1,R2,fun identity/1)),
281 Test42 = assertEquals({ok,State},
282   at_server:query_t(Pid1,R3,fun identity/1)),
283
284 ok = at_server:update_t(Pid1,R3,fun removeEven/1),
285 timer:sleep(?SLEEP_TIME),
286 Test43 = assertEquals({ok,removeEven(State)},
287   at_server:query_t(Pid1,R3,fun identity/1)),
288 Test4 = areTrue([Test41, Test42, Test43]),
289
290 %% 5. Test what happens with a wrong ref_id
291 {ok, AllPids} = at_server:get_pids(Pid1),
292 Test51 = assertEquals(true, areProcessesAlive(AllPids)),
293 ok = at_server:update_t(Pid1,wrong_ref,fun removeEven/1),
294 timer:sleep(?SLEEP_TIME),
295 Test52 = assertEquals(true, areProcessesAlive(AllPids)),
296 Test53 = assertEquals({ok,removeEven(State)},
297   at_server:query_t(Pid1,R3,fun identity/1)),
298 Test5 = areTrue([Test51, Test52, Test53]),
299
300 %% Clean up
301 {ok,State} = at_server:stop(Pid1),
302
303 areTrue([Test1, Test2, Test3, Test4, Test5]).
304
305 %% Test commit_t/2
306 testCommit_t() =
307   %% Init test values
308   StateA = [1,2,3,4,5,6,7,8,9,10],
309   StateB = removeEven(StateA),
310   {ok,Pid1} = at_server:start(StateA),
311   {ok,R1} = at_server:begin_t(Pid1),
312   timer:sleep(?SLEEP_TIME),
313
314   %% 1. Test that after a commit without first doing a update the state is still the same
315   %% And that it is still treated as a commit, ie the process is aborted
316   Test11 = assertEquals({ok,StateA},
317     at_server:doquery(Pid1, fun identity/1)),
318   Test12 = assertEquals(ok, at_server:commit_t(Pid1,R1)),
319   Test13 = assertEquals({ok,StateA},
320     at_server:doquery(Pid1, fun identity/1)),
321   Test1 = areTrue([Test11, Test12, Test13]),
322
323   %% 2. Test that after a commit the transactions are all aborted
324   Test21 = assertEquals(true, isAborted(Pid1,R1)),
325   Test22 = assertEquals(aborted, at_server:commit_t(Pid1,R1)),
326   Test23 = assertEquals(true, isAborted(Pid1,R1)),
327   Test2 = areTrue([Test21, Test22, Test23]),
328
329   %% 3. Test that the state changes to the correct value after a commit and update
330   {ok,R2} = at_server:begin_t(Pid1),
331   Test31 = assertEquals({ok,StateA},
332     at_server:doquery(Pid1, fun identity/1)),
333   ok = at_server:update_t(Pid1,R2, fun removeEven/1),
334   timer:sleep(?SLEEP_TIME),
335   ok = at_server:commit_t(Pid1,R2),

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

336 Test32 = assertEquals({ok,StateB},
337   at_server:doquery(Pid1, fun identity/1)),
338 Test3 = areTrue([Test31, Test32]),
339
340 %% 4. Test if we try to commit after the update function have failed
341 {ok,R3} = at_server:begin_t(Pid1),
342 ok = at_server:update_t(Pid1,R3, fun onlyEmpty/1),
343 Test4 = assertEquals(aborted, at_server:commit_t(Pid1,R3)),
344
345 %% 5. Test that we can have several different transactions going at one time
346 %% And that all are aborted when one is committed
347 {ok,R4} = at_server:begin_t(Pid1),
348 {ok,R5} = at_server:begin_t(Pid1),
349 {ok,R6} = at_server:begin_t(Pid1),
350 {ok,R7} = at_server:begin_t(Pid1),
351 timer:sleep(?SLEEP_TIME),
352
353 Mult2 = fun(NS) -> lists:map(fun(N) -> N*2 end,NS) end,
354 Mult4 = fun(NS) -> lists:map(fun(N) -> N*4 end,NS) end,
355 Mult8 = fun(NS) -> lists:map(fun(N) -> N*8 end,NS) end,
356
357 ok = at_server:update_t(Pid1,R4, Mult2),
358 ok = at_server:update_t(Pid1,R5, Mult4),
359 ok = at_server:update_t(Pid1,R6, fun onlyEmpty/1),
360 ok = at_server:update_t(Pid1,R7, Mult8),
361 timer:sleep(?SLEEP_TIME),
362
363 Test51 = assertEquals({ok,Mult2(StateB)},
364   at_server:query_t(Pid1,R4,fun identity/1)),
365 Test52 = assertEquals({ok,Mult4(StateB)},
366   at_server:query_t(Pid1,R5,fun identity/1)),
367 Test53 = assertEquals(aborted,
368   at_server:query_t(Pid1,R6,fun identity/1)),
369 Test54 = assertEquals({ok,Mult8(StateB)},
370   at_server:query_t(Pid1,R7,fun identity/1)),
371
372 ok = at_server:commit_t(Pid1,R5),
373 StateC = Mult4(StateB),
374 Test55 = assertEquals({ok,StateC},
375   at_server:doquery(Pid1, fun identity/1)),
376 Test56 = assertEquals(true, isAborted(Pid1,R4)),
377 Test57 = assertEquals(true, isAborted(Pid1,R5)),
378 Test58 = assertEquals(true, isAborted(Pid1,R6)),
379 Test59 = assertEquals(true, isAborted(Pid1,R7)),
380 Test5 = areTrue([Test51,Test52,Test53,Test54,Test55,Test56,
381   Test57,Test58,Test59]),
382
383 %% 6. Test with wrong ref
384 Test6 = assertEquals(aborted, at_server:commit_t(Pid1,wrong_ref)),
385
386 %% Clean up
387 {ok,StateC} = at_server:stop(Pid1),
388
389 areTrue([Test1, Test2, Test3, Test4, Test5, Test6]).
390
391 %% -----
392 %% ----- EXT API -----
393
394 %% Tests abort/2
395 testAbort() ->
396   %% Init test data
397   State = abcdef,

```



```

398 {ok,Pid1} = at_server:start(State),
399 {ok,R1} = at_server:begin_t(Pid1),
400 {ok,R2} = at_server:begin_t(Pid1),
401 {ok,R3} = at_server:begin_t(Pid1),
402 {ok,R4} = at_server:begin_t(Pid1),
403 {ok,R5} = at_server:begin_t(Pid1),
404 timer:sleep(?SLEEP_TIME),
405
406 %% 1. Test that the transaction is aborted
407 Test11 = assertEquals(aborted, at_extapi:abort(Pid1,R1)),
408 Test12 = assertEquals(true, isAborted(Pid1,R1)),
409 Test1 = areTrue([Test11, Test12]),
410
411 %% 2. Test what happens if calling aborted again
412 Test2 = assertEquals(aborted, at_extapi:abort(Pid1,R1)),
413
414 %% 3. Test that several can be aborted
415 Test31 = assertEquals(aborted, at_extapi:abort(Pid1,R2)),
416 Test32 = assertEquals(aborted, at_extapi:abort(Pid1,R3)),
417 Test3 = areTrue([Test31, Test32]),
418
419 %% 4. Test that no one else is affected when one is being aborted
420 Test41 = assertEquals({ok,State}, at_server:query_t(Pid1,R4,fun identity/1)),
421 ok = at_server:update_t(Pid1,R5,fun atom_to_list/1),
422 timer:sleep(?SLEEP_TIME),
423 Test42 = assertEquals({ok,atom_to_list(State)}, at_server:query_t(Pid1,R5,fun identity/1)),
424 Test4 = areTrue([Test41, Test42]),
425
426 %% 5. Test what happens with a unknown ref id
427 WrongRef = make_ref(),
428 Test5 = assertEquals(aborted, at_extapi:abort(Pid1,WrongRef)),
429
430 %% Clean up
431 {ok,State} = at_server:stop(Pid1),
432
433 areTrue([Test1, Test2, Test3, Test4, Test5]).
434
435 %% Tests tryUpdate/2
436 testTryUpdate() =>
437   %% Init test data
438   StateA = [1,2,3,4,5,6],
439   StateB = removeEven(StateA),
440   {ok,Pid1} = at_server:start(StateA),
441   timer:sleep(?SLEEP_TIME),
442
443   %% 1. Test that if no one else is doing a transaction we get our update through
444   Test11 = assertEquals(ok, at_extapi:tryUpdate(Pid1,fun identity/1)),
445   Test12 = assertEquals({ok,StateA}, at_server:doquery(Pid1, fun identity/1)),
446   Test13 = assertEquals(ok, at_extapi:tryUpdate(Pid1,fun removeEven/1)),
447   Test14 = assertEquals({ok,StateB}, at_server:doquery(Pid1, fun identity/1)),
448   Test1 = areTrue([Test11, Test12, Test13, Test14]),
449
450   %% 2. Test that if the function fails, no update happens and we get error returned
451   Test21 = assertEquals(error, at_extapi:tryUpdate(Pid1,fun onlyEmpty/1)),
452   Test22 = assertEquals({ok,StateB}, at_server:doquery(Pid1, fun identity/1)),
453   Test2 = areTrue([Test21, Test22]),
454
455   %% 3. Test that if others is doing a transaction they get aborted
456   {ok,R1} = at_server:begin_t(Pid1),
457   {ok,R2} = at_server:begin_t(Pid1),
458   {ok,R3} = at_server:begin_t(Pid1),
459   timer:sleep(?SLEEP_TIME),

```

```

460 ok = at_server:update_t(Pid1,R2,fun onlyEmpty/1),
461 ok = at_server:update_t(Pid1,R3,fun removeEven/1),
462 timer:sleep(?SLEEP_TIME),
463
464 Test31 = assertEquals(ok, at_extapi:tryUpdate(Pid1,fun identity/1)),
465 Test32 = assertEquals(true, isAborted(Pid1,R1)),
466 Test33 = assertEquals(true, isAborted(Pid1,R2)),
467 Test34 = assertEquals(true, isAborted(Pid1,R3)),
468 Test3 = areTrue([Test31, Test32, Test33, Test34]),
469
470 %% 4. Test that if someone commits while we are trying to update we get aborted
471 %% -> Really hard to do due to the implementation of tryUpdate/2
472 {ok,R} = at_server:begin_t(Pid1),
473 ok = at_server:update_t(Pid1,R,fun(_) ->
474     T = expensivel(500),
475     at_server:commit_t(Pid1,R),
476     T end),
477 Test4 = assertNotEquals(aborted,
478     at_extapi:tryUpdate(Pid1,fun(_) -> expensivel(3500) end)),
479
480 %% Clean up
481 {ok,3500} = at_server:stop(Pid1),
482
483 areTrue([Test1, Test2, Test3, Test4]).
484
485 %% Tests ensureUpdate/2
486 testEnsureUpdate() ->
487     %% Init test data
488     StateA = [1,2,3,4,5,6],
489     StateB = removeEven(StateA),
490     {ok,Pid} = at_server:start(StateA),
491     timer:sleep(?SLEEP_TIME),
492
493     %% 1. Test that if we are the only one here we get our update through
494     Test11 = assertEquals(ok, at_extapi:ensureUpdate(Pid,fun identity/1)),
495     Test12 = assertEquals({ok,StateA}, at_server:doquery(Pid,fun identity/1)),
496     Test13 = assertEquals(ok, at_extapi:ensureUpdate(Pid,fun removeEven/1)),
497     Test14 = assertEquals({ok,StateB}, at_server:doquery(Pid,fun identity/1)),
498     Test1 = areTrue([Test11, Test12, Test13, Test14]),
499
500     %% 2. Test that if the function fails we get error and nothing is updated
501     Test21 = assertEquals(error, at_extapi:ensureUpdate(Pid,fun onlyEmpty/1)),
502     Test22 = assertEquals({ok,StateB}, at_server:doquery(Pid,fun identity/1)),
503     Test2 = areTrue([Test21, Test22]),
504
505     %% 3. Test that if someone commits while we are trying to update we still get our
506     %% commit through on the original state and the other is rolled backed
507     {ok,R} = at_server:begin_t(Pid),
508     ok = at_server:update_t(Pid,R,fun(_) ->
509         expensivel(500),
510         at_server:commit_t(Pid,R) end),
511     Test3 = assertEquals(ok,
512         at_extapi:ensureUpdate(Pid,fun(_) -> expensivel(3500) end)),
513
514     %% Clean up
515     {ok,3500} = at_server:stop(Pid),
516
517     areTrue([Test1, Test2, Test3]).
518
519 %% Tests choiceUpdate/3
520 testChoiceUpdate() ->
521     %% Init test data

```

```

522 Val_listA = [1],
523 Val_listB = [a,2,c],
524 %% TODO timeout error happens if 500 is set to 5000+
525 Val_listC = [500,1],
526 Val_listD = [a,b,c],
527
528 Add = fun(State,E) -> lists:map(fun(N) -> N+E end,State) end,
529 StateA = [1,2,3,4,5],
530 StateB = Add(StateA,lists:nth(1,Val_listA)),
531 StateC = Add(StateB,lists:nth(2,Val_listB)),
532 StateD = lists:nth(2,Val_listC),
533
534 {ok,Pid} = at_server:start(StateA),
535 timer:sleep(?SLEEP_TIME),
536
537 %% 1. Test that if only one then that gets through
538 Test11 = assertEquals(ok, at_extapi:choiceUpdate(Pid,Add,Val_listA)),
539 Test12 = assertEquals({ok,StateB}, at_server:doquery(Pid,fun identity/1)),
540 Test1 = areTrue([Test11, Test12]),
541
542 %% 2. Test that if all fail except one, then that gets through
543 Test21 = assertEquals(ok, at_extapi:choiceUpdate(Pid,Add,Val_listB)),
544 Test22 = assertEquals({ok,StateC}, at_server:doquery(Pid,fun identity/1)),
545 Test2 = areTrue([Test21, Test22]),
546
547 %% 3. Test that a shorter function will be the one to come through rather than a long function
548 %% Although, this cannot be guaranteed
549 Test31 = assertEquals(ok, at_extapi:choiceUpdate(Pid,fun expensive/2,Val_listC)),
550 Test32 = assertEquals({ok,StateD}, at_server:doquery(Pid,fun identity/1)),
551 Test3 = areTrue([Test31, Test32]),
552
553 %% 4. Test that if someone else commits before any of us we get aborted when trying to commit
554 {ok,R} = at_server:begin_t(Pid),
555 Test41 = assertEquals(aborted,
556   at_extapi:choiceUpdate(Pid,fun(_,_) ->
557     at_server:commit_t(Pid,R)
558   and Val_listA)),
559 Test42 = assertEquals({ok,StateD}, at_server:doquery(Pid,fun identity/1)),
560 Test4 = areTrue([Test41, Test42]),
561
562 %% 5. Test that if all fail then error is returned
563 Test5 = assertEquals(error, at_extapi:choiceUpdate(Pid,Add,Val_listD)),
564
565 %% 6. Test that if the list is empty
566 Test6 = assertEquals(error, at_extapi:choiceUpdate(Pid,Add,[])),
567
568 %% Clean up
569 {ok,StateD} = at_server:stop(Pid),
570
571 areTrue([Test1, Test2, Test3, Test4, Test5, Test6]).
572
573 %%%
574 %%%
575 %%% Helper Functions
576 %%%
577
578 areTrue([]) ->
579   false;
580 areTrue(List) ->
581   lists:foldl(fun(A,B) -> A andalso B end,true,List).
582
583 assertEquals(A,B) ->

```

```

584     A == B.
585
586 assertNotEquals(A,B) ->
587     A /= B.
588
589 isAborted(Pid,R) ->
590     A1 = aborted == at_server:query_t(Pid,R, fun identity/1),
591     A2 = aborted == at_server:commit_t(Pid,R),
592     A1 andalso A2.
593
594 % Returns true if the given Pid is a running process
595 % otherwise false.
596 isProcessAlive(Pid) ->
597     case process_info(Pid) of
598     undefined ->
599         false;
600     _ -> true
601     end.
602
603 isProcessDead(Pid) ->
604     not(isProcessAlive(Pid)).
605
606 areProcessesAlive(Pids) ->
607     lists:foldl(fun(P,B) -> isProcessAlive(P) andalso B end,true,Pids).
608
609 areProcessesDead(Pids) ->
610     lists:foldl(fun(P,B) -> (isProcessDead(P)) andalso B end,true,Pids).
611
612 begin_transactions(A,N) ->
613     case N > 0 of
614     true ->
615         {ok, R} = at_server:begin_t(A),
616         [R] begin_transactions(A,N-1)
617     false -> []
618     end.
619
620 %%%
621 %%% Update Functions
622 %%%
623
624 expensive(_,Time) ->
625     receive
626     after
627     Time -> Time
628     end.
629
630 expensive1(Time) ->
631     expensive(Time,Time).
632
633 identity(X) ->
634     X.
635
636 onlyEmpty([]) ->
637     [].
638
639 mapToA(_) ->
640     "A".
641
642 mapMult2(Ns) ->
643     lists:map(fun(X) -> X*2 end,Ns).
644
645 removeEven(X) ->

```

```
646 | lists : filter (fun (N) -> N rem 2 /= 0 end, X).
```

9.4 Gpx.hs

```
1 module Gpx where
2
3 type ViewName = String
4 type ColourName = String
5 type Frame = [GpxInstr]
6 type Animation = ([[ViewName, Integer, Integer]], [Frame])
7 data GpxInstr = DrawRect Integer Integer Integer Integer ViewName ColourName
8             | DrawCirc Integer Integer Integer Integer ViewName ColourName
9             deriving (Eq, Show)
```

9.5 SalsaAst.hs

```
1 module SalsaAst where
2
3 type Program = [DefCom]
4 data DefCom = Def Definition
5             | Com Command
6             deriving (Show, Eq)
7 data Definition = Def Ident Expr Expr
8               | Rectang Ident Expr Expr Expr Expr Colour
9               | Circle Ident Expr Expr Expr Colour
10              | View Ident
11              | Group Ident [Ident]
12              deriving (Show, Eq)
13 data Command = Mov Ident Pos
14              | At Command Ident
15              | Par Command Command
16              deriving (Show, Eq)
17 data Pos = Abs Expr Expr
18          | Rel Expr Expr
19          deriving (Show, Eq)
20 data Expr = Plus Expr Expr
21          | Minus Expr Expr
22          | Const Integer
23          | Xproj Ident
24          | Yproj Ident
25          deriving (Show, Eq)
26 data Colour = Blue | Plum | Red | Green | Orange
27             deriving (Show, Eq)
28 type Ident = String
```

9.6 SalsaInterp.hs

```
1 -----
2 — Student name: Arni Asgeirsson
3 — Student KU-id: lwf986
4 -----
5 module SalsaInterp
6   (Position, interpolate, runProg)
7 where
8
```

```

9 import SalsaAst
10 import Gpx
11 import qualified Data.Map as M
12 import qualified Data.Maybe as Mb
13 import qualified Control.Monad as Mo
14
15 ----- The interface -----
16
17
18
19 type Position = (Integer, Integer)
20
21 interpolate :: Integer -> Position -> Position -> [Position]
22 interpolate 0 _ _ = []
23 interpolate framerate (x1,y1) pe@(x2,y2) =
24   let
25     rate = 100 `div` framerate
26     xdis = (x2 - x1) * rate
27     ydis = (y2 - y1) * rate
28   in
29     [(x,y) | i <- [1..(framerate-1)], x <- [x1+((xdis*i) `div` 100)],
30       y <- [y1+((ydis*i) `div` 100)]] ++ [pe]
31
32 runProg :: Integer -> Program -> Animation
33 runProg n p =
34   let (_,(_,anim)) = runSalsa (createEmptyContext n)
35   in (Mo.join (Salsa $ \con -> (mapM defCom p, con)))
36     anim
37
38 ----- Context -----
39
40 ----- Data Structure -----
41
42
43
44 data Context = Context ConEnvironment State
45   deriving (Show)
46 type ConEnvironment = Environment [Ident] Integer
47 type Environment = M.Map Ident Definition
48 type State = M.Map Ident [(Ident, Position)]
49
50 ----- Working on the DS -----
51
52
53 createEmptyContext :: Integer -> Context
54 createEmptyContext n = Context (M.empty, [], n) M.empty
55
56 lookupViews :: Ident -> Environment -> [Ident]
57 lookupViews key env =
58   case lookupKey key env of
59     (Viewdef view _ _) ->
60       [view]
61     (Group _ views) ->
62       views
63     _ ->
64       error $ "Tried to look up " ++ key ++ " found something not expected"
65
66 bindCommand :: Ident -> [(Ident, Position)] -> State -> State
67 bindCommand = M.insert
68
69 addToState :: (State -> State) -> Context -> State
70 addToState f (Context _ state) = f state

```

```

71 |
72 | bindDefinition :: Ident -> Definition -> Environment -> Environment
73 | bindDefinition = M.insert
74 |
75 | addToEnvironment :: (Environment -> Environment) -> Context -> Context
76 | addToEnvironment f (Context (env,active,fr) state) =
77 |   Context (f env,active,fr) state
78 |
79 | updateActiveViews :: [Ident] -> Context -> Context
80 | updateActiveViews views (Context (env,_,fr) state) =
81 |   Context (env,views,fr) state
82 |
83 | placeShapeInActiveViews :: Definition -> Context -> Context
84 | placeShapeInActiveViews (Rectangle id_ (Const x) (Const y) _ _ _) con =
85 |   placeShapeHelper id_ (x,y) con
86 | placeShapeInActiveViews (Circle id_ (Const x) (Const y) _ _ _) con =
87 |   placeShapeHelper id_ (x,y) con
88 | placeShapeInActiveViews _ _ = error "Trying to place something that is not a shape in the active views"
89 |
90 | placeShapeHelper :: Ident -> Position -> Context -> Context
91 | placeShapeHelper id_ pos con =
92 |   let (Context (env,active,fr) state) = con
93 |       positions = map (\view_id -> (view_id,pos)) active
94 |       newState = M.insert id_ positions state
95 |   in
96 |     Context (env,active,fr) newState
97 |
98 | ----- Working on Animation -----
99 |
100 | goToNextFrame :: Animation -> Animation
101 | goToNextFrame (views, frames) = (views,frames++[[[]]])
102 |
103 | addInstructions :: [GpxInstr] -> Animation -> Animation
104 | addInstructions new_instr (view, frames) =
105 |   let (rest,[curframe]) = getLast frames
106 |   in
107 |     (views, rest++[curframe++new_instr])
108 |
109 | placeShapeInCurrentFrame :: Definition -> [ViewName] -> Animation -> Animation
110 | placeShapeInCurrentFrame (Rectangle _ (Const x) (Const y) (Const w) (Const h) colour) active anim =
111 |   placeShapeFrameHelper (DrawRect x y w h) colour active anim
112 | placeShapeInCurrentFrame (Circle _ (Const x) (Const y) (Const r) colour) active anim =
113 |   placeShapeFrameHelper (DrawCirc x y r) colour active anim
114 | placeShapeInCurrentFrame _ _ _ = error "Trying to place something that is not a shape in the current frame"
115 |
116 | placeShapeFrameHelper :: (a -> ColourName -> GpxInstr)
117 |   -> Colour -> [a] -> Animation -> Animation
118 | placeShapeFrameHelper shape colour active anim =
119 |   let col = evalColour colour
120 |       new_instr = map f active
121 |       f view = shape view col
122 |   in
123 |     addInstructions new_instr anim
124 |
125 | addViewToAnimation :: (ViewName,Integer,Integer) -> Animation -> Animation
126 | addViewToAnimation v (views, frames) = (views++[v],frames)
127 |
128 | -----
129 | ----- Monads -----
130 | -----
131 | -----
132 | -----

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

133 ----- Monads Types -----
134
135
136 ----- SalsaCommand -----
137
138 newtype SalsaCommand a = SalsaCommand {runSC :: Context -> (a,State)}
139
140 instance Monad SalsaCommand where
141   return k = SalsaCommand $ \(Context _ state) -> (k,state)
142   m >>= f = SalsaCommand $ \c ->
143     let (Context e _) = c
144         (a,state1) = runSC m c
145         m1 = f a
146     in
147       runSC m1 (Context e state1)
148
149 ----- Salsa -----
150
151 data Salsa a = Salsa ((Context,Animation) -> (a,(Context,Animation)))
152
153 instance Monad Salsa where
154   return k = Salsa $ \state -> (k,state)
155   (Salsa a1) >>= f = Salsa $ \state0 -> let (r,state1) = a1 state0
156                                         (Salsa a2) = f r
157                                         in
158                                           a2 state1
159
160 ----- Accessors -----
161
162 ----- SalsaCommand -----
163
164
165 askCmd :: SalsaCommand Context
166 askCmd = SalsaCommand $ \con@(Context _ s) -> (con,s)
167
168 updateState :: (Context -> State) -> SalsaCommand ()
169 updateState f = SalsaCommand $ \con -> (((), f con)
170
171 ----- Salsa -----
172
173 askCont :: Salsa Context
174 askCont = Salsa $ \s@(con,_) -> (con,s)
175
176 runSalsa :: Context -> Salsa a -> (a,(Context,Animation))
177 runSalsa con (Salsa m) = m (con,([],[]))
178
179 updateContext :: (Context -> Context) -> Salsa ()
180 updateContext f = Salsa $ \con,anim -> (((), (f con,anim))
181
182 updateAnimation :: (Animation -> Animation) -> Salsa ()
183 updateAnimation f = Salsa $ \con,anim -> (((), (con, f anim))
184
185 ----- Interpret Functions -----
186
187
188 ----- SalsaCommand -----
189
190
191 command :: Command -> SalsaCommand ()
192 command (Move ids point) = do
193   con <- askCmd
194   let (Context (_,active,_) state) = con

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder


```

195 (h:_) <- mapM (\x ->
196     do
197         let list = lookupKey x state
198         newList <- mapM (setNextPosition active point) list
199         updateState (addToState (bindCommand x newList))
200     ) ids
201 return h -- Dummy return
202 command (At cmd id_) = do
203     (Context (env, active, _) state) <- askCmd
204     let vs = lookupViews id_ env
205     (tmp_state, mapping) <- setTmpActiveViews state active vs
206     updateState $ const tmp_state
207     command cmd
208     (Context (_, _, _) state1) <- askCmd
209     next_state <- revertActiveViews state1 mapping
210     updateState $ const next_state
211 command (Par cmd1 cmd2) = do
212     command cmd1
213     command cmd2
214
215 ----- Salsa -----
216
217 defCom :: DefCom -> Salsa ()
218 defCom (Def def) = definition def
219 defCom (Com cmd) = do
220     con <- askCont
221     let (s, state) = runSC (command cmd) con
222     (Context e s) = con
223     newcon = Context e state
224     instr <- compareStates s state
225     updateContext $ const newcon
226     updateAnimation $ addInstructions instr
227     updateAnimation toNextFrame
228
229 definition :: Definition -> Salsa ()
230 definition (Viewdef id_ x y) = do
231     valx <- evalExpr x askCont
232     valy <- evalExpr y askCont
233     if valx < 0 || valy < 0 then
234         error "A view cannot be defined with a negative width or height!"
235     else do
236         updateContext (addToEnvironment (bindDefinition id_ $ Viewdef id_ (Const valx) (Const valy)))
237         updateContext (updateActiveViews [id_])
238         updateAnimation (addViewToAnimation (id_, valx, valy))
239 definition (View id_) = do
240     (Context (env, _, _) _) <- askCont
241     let views = lookupViews id_ env
242     updateContext (updateActiveViews views)
243 definition (Group id_ views) = do
244     updateContext (addToEnvironment (bindDefinition id_ $ Group id_ views))
245     updateContext (updateActiveViews views)
246 definition (Rectangle id_ x y w h colour) = do
247     valx <- evalExpr x askCont
248     valy <- evalExpr y askCont
249     valw <- evalExpr w askCont
250     valh <- evalExpr h askCont
251     if valw < 0 || valh < 0 then
252         error "A rectangle cannot be defined with a negative width or height!"
253     else
254         let newRect = Rectangle id_ (Const valx) (Const valy) (Const valw) (Const valh) colour
255         in do
256             updateContext (addToEnvironment (bindDefinition id_ newRect))

```

Add WeChat powcoder

```

319   return $ M.fromList new_state
320
321 setNextPosition :: Eq t => [t] -> Pos ->
322   (t, Position) -> SalsaCommand (t, Position)
323 setNextPosition active point (view,pos) =
324   if view 'elem' active
325   then do
326     next <- evalNextPoint point pos
327     return (view,next)
328   else
329     return (view,pos)
330
331 getLowestPosition :: Position -> (ViewName,Position) -> Position
332 getLowestPosition (l_x,l_y) (_,(x,y)) = let
333   n_x = if x < l_x then x else l_x
334   n_y = if y < l_y then y else l_y
335   in
336   (n_x,n_y)
337
338 ----- Salsa -----
339
340 -- This function assumes that the set of keys in old is the same as in new
341 compareStates :: State -> State -> Salsa [GpxInstr]
342 compareStates old_s new_s = let
343   s1 = M.toList old_s
344   s2 = M.toList new_s
345   in
346   do
347     l <- mapM (zip s1) s2
348     return $ concat l
349   where
350     fg old (ident,new_positions) =
351       case lookup ident old of
352         Just old_positions ->
353           do
354             (Context (env,_,framerate) _) <- askCont
355             l <- generateInstructions (lookupKey ident env) framerate
356             old_positions new_positions
357             return $ concat l
358         Nothing ->
359           error $ ident++" did not exist in the new state"
360
361 -- It is assumed that every viewname in old_p.. must also appear in new_p..
362 generateInstructions :: Definition -> Integer -> [(ViewName, Position)] -> [(ViewName, Position)] ->
363 generateInstructions s fr oldPos =
364   mapM (genInstrHelper s fr oldPos)
365   where
366     genInstrHelper shape framerate old_positions (viewName, new_pos) =
367       case lookup viewName old_positions of
368         Just old_pos ->
369           if old_pos == new_pos
370           then return []
371           else mapM (positionToInstr shape viewName) (interpolate framerate old_pos new_pos)
372         Nothing ->
373           error $ viewName++" did not exist in the list of old_positions"
374
375 positionToInstr :: Definition -> ViewName -> Position -> Salsa GpxInstr
376 positionToInstr (Rectangle _ _ _ expw exp h expcol) viewName (x,y) = do
377   w <- evalExpr expw askCont
378   h <- evalExpr exp h askCont
379   return $ DrawRect x y w h viewName (evalColour expcol)
380 positionToInstr (Circle _ _ _ expr expcol) viewName (x,y) = do

```

```

381   r <- evalExpr expr askCont
382   return $ DrawCirc x y r viewName (evalColour expcol)
383 positionToInstr _ _ _ = error "Trying to create instructions from something that is not a shape"
384
385 evalExpr :: Monad m => Expr -> m Context -> m Integer
386 evalExpr (Const int) _ = return int
387 evalExpr (Plus exp1 exp2) askf = do
388   x <- evalExpr exp1 askf
389   y <- evalExpr exp2 askf
390   return $ x+y
391 evalExpr (Minus exp1 exp2) askf = do
392   x <- evalExpr exp1 askf
393   y <- evalExpr exp2 askf
394   return $ x-y
395 evalExpr (Xproj ident) askf = do
396   (Context _ state) <- askf
397   let max_i = toInteger(maxBound :: Int)
398       (x,_) = foldl getLowestPosition (max_i,max_i) $ lookupKey ident state
399   return x
400 evalExpr (Yproj ident) askf = do
401   (Context _ state) <- askf
402   let max_i = toInteger(maxBound :: Int)
403       (_,y) = foldl getLowestPosition (max_i,max_i) $ lookupKey ident state
404   return y
405
406 -----
407 ----- Helper Functions -----
408 -----
409
410 lookupKey :: Ord a => a -> M.Map a b -> b
411 lookupKey key env =
412   Mb.fromMaybe (error "Tried to look up unknown key ")
413   (M.lookup key env)
414
415
416 removeDouble :: Eq a => [a] -> [a] -> [a]
417 removeDouble [] _ = []
418 removeDouble _ [] = []
419 removeDouble (x:xs) ys =
420   if x `elem` ys
421   then removeDouble xs ys
422   else x:removeDouble xs ys
423
424 evalColour :: Colour -> ColourName
425 evalColour Blue = "blue"
426 evalColour Plum = "plum"
427 evalColour Red = "red"
428 evalColour Green = "green"
429 evalColour Orange = "orange"
430
431 getLast :: [a] -> ([a],[a])
432 getLast [] = ([],[a])
433 getLast (x:[]) = ([],[x])
434 getLast (x:xs) = let (rest,last_) = getLast xs
435                   in
436                   (x:rest ,last_)

```

9.7 SalsaParser.hs

```

2  — Student name: Arni Asgeirsson
3  — Student KU-id: lwf986
4
5  module SalsaParser (parseString, parseFile, Error (..)) where
6
7  import SalsaAst
8  import Text.ParserCombinators.ReadP
9
10
11  ————— The interface —————
12
13
14  data Error = NoParsePossible String
15             | AmbiguousGrammar [(Program, String)]
16             | UnexpectedRemainder Program String
17             deriving (Eq, Show)
18
19  parseString :: String -> Either Error Program
20  parseString = parse runParser
21
22  parseFile :: FilePath -> IO (Either Error Program)
23  parseFile filename = do
24    content <- readFile filename
25    return $ parseString content
26
27
28  ————— The parser functions —————
29
30
31  ————— Definitions —————
32
33  identarr :: String
34  identarr = ['A'..'z'] ++ ['a'..'z'] ++ ['0'..'9'] ++ ["_"]
35
36  reservedWords :: [String]
37  reservedWords = ["viewdef", "rectangle", "circle", "group", "view"]
38
39  colourNames :: [String]
40  colourNames = ["blue", "blun", "red", "green", "orange"]
41
42  ————— Top-Level parsers —————
43
44  parse :: ReadP Program -> String -> Either Error Program
45  parse parser s =
46    case readP_to_S parser s of
47      [(result, "")] -> Right result
48      [(result, unparsed)] -> Left $ UnexpectedRemainder result unparsed
49      [] -> Left $ NoParsePossible s
50      results -> Left $ AmbiguousGrammar results
51
52  runParser :: ReadP Program
53  runParser = do
54    skipSpaces
55    p <- pProgram
56    skipSpaces
57    eof
58    return p
59
60  ————— Grammar Parsers —————
61
62  pProgram :: ReadP Program
63  pProgram = pDefComs

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
64 pDefComs :: ReadP [DefCom]
65 pDefComs = do
66   dc <- pDefCom
67   dcs <- pDefComs'
68   return $ dc:dcs
69
70
71 pDefComs' :: ReadP [DefCom]
72 pDefComs' = pDefComs +++ return []
73
74 pDefCom :: ReadP DefCom
75 pDefCom = do
76   cmd <- pCommand
77   return $ Com cmd
78 +++
79   do
80     def <- pDefinition
81     return $ Def def
82
83 pDefinition :: ReadP Definition
84 pDefinition = hViewdef +++ hRectangle +++ hCircle +++ hView +++ hGroup
85
86 hViewdef :: ReadP Definition
87 hViewdef = do
88   stringT "viewdef"
89   vid <- pVIdent
90   exp1 <- pExpr
91   exp2 <- pExpr
92   return $ Viewdef vid exp1 exp2
93
94 hRectangle :: ReadP Definition
95 hRectangle = do
96   stringT "rectangle"
97   sid <- pSIdent
98   exp1 <- pExpr
99   exp2 <- pExpr
100  exp3 <- pExpr
101  exp4 <- pExpr
102  col <- pColour
103  return $ Rectangle sid exp1 exp2 exp3 exp4 col
104
105 hCircle :: ReadP Definition
106 hCircle = do
107   stringT "circle"
108   sid <- pSIdent
109   exp1 <- pExpr
110   exp2 <- pExpr
111   exp3 <- pExpr
112   col <- pColour
113   return $ Circle sid exp1 exp2 exp3 col
114
115 hView :: ReadP Definition
116 hView = do
117   stringT "view"
118   vid <- pVIdent
119   return $ View vid
120
121 hGroup :: ReadP Definition
122 hGroup = do
123   stringT "group"
124   vid <- pVIdent
125   vids <- bracks '[' pVIdents ']'
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

126 |   return $ Group vid vids
127 |
128 | pCommand :: ReadP Command
129 | pCommand = do
130 |   cmd2 <- pCommand2
131 |   pCommand' cmd2
132 |
133 | pCommand' :: Command -> ReadP Command
134 | pCommand' iV = do
135 |   stringT "||"
136 |   cmd2 <- pCommand2
137 |   pCommand' $ Par iV cmd2
138 |   +++
139 |   return iV
140 |
141 | pCommand2 :: ReadP Command
142 | pCommand2 = do
143 |   cmd3 <- pCommand3
144 |   pCommand2' cmd3
145 |
146 | pCommand2' :: Command -> ReadP Command
147 | pCommand2' iV = do
148 |   charT '@'
149 |   vid <- pVIdent
150 |   pCommand2' $ At iV vid
151 |   +++
152 |   return iV
153 |
154 | pCommand3 :: ReadP Command
155 | pCommand3 = do
156 |   sids <- pSIdent
157 |   stringT "=>"
158 |   pos <- pPos
159 |   return $ Move sids pos
160 |   +++
161 |   bracks '{' pCommand '}'
162 |
163 | pVIdents :: ReadP [Ident]
164 | pVIdents = do
165 |   vid <- pVIdent
166 |   vids' <- pVIdents'
167 |   return $ vid:vids'
168 |
169 | pVIdents' :: ReadP [Ident]
170 | pVIdents' = pVIdents +++ return []
171 |
172 | pSIdents :: ReadP [Ident]
173 | pSIdents = do
174 |   sid <- pSIdent
175 |   sids' <- pSIdents'
176 |   return $ sid:sids'
177 |
178 | pSIdents' :: ReadP [Ident]
179 | pSIdents' = pSIdents +++ return []
180 |
181 | pPos :: ReadP Pos
182 | pPos =
183 |   bracks '(' (hMiddle Abs) ')'
184 |   +++
185 |   do
186 |     charT '+'
187 |     bracks '(' (hMiddle Rel) ')'

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
188
189 hMiddle :: (Expr -> Expr -> b) -> ReadP b
190 hMiddle c = do
191   exp1 <- pExpr
192   charT ','
193   exp2 <- pExpr
194   return $ c exp1 exp2
195
196 pExpr :: ReadP Expr
197 pExpr = do
198   prim <- pPrim
199   pExpr' prim
200
201 pExpr' :: Expr -> ReadP Expr
202 pExpr' iV = do
203   exp_ <- pOp iV
204   pExpr' exp_
205   +++
206   return iV
207
208 pOp :: Expr -> ReadP Expr
209 pOp iV = do
210   charT '+'
211   prim <- pPrim
212   return $ Plus iV prim
213   +++
214   do
215     charT '-'
216     prim <- pPrim
217     return $ Minus iV prim
218
219 pPrim :: ReadP Expr
220 pPrim = do
221   int <- pInteger
222   return $ Const int
223   +++
224   bracks '(' pExpr ')'
225   +++
226   do
227     sid <- pSIdent
228     charT '.'
229     pProj sid
230
231 pProj :: Ident -> ReadP Expr
232 pProj iV = do
233   charT 'x'
234   return $ Xproj iV
235   +++
236   do
237     charT 'y'
238     return $ Yproj iV
239
240 pColour :: ReadP Colour
241 pColour = do
242   stringT "blue"
243   return Blue
244   +++
245   do
246     stringT "plum"
247     return Plum
248   +++
249   do
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder


```

250   stringT "red"
251   return Red
252   +++
253   do
254     stringT "green"
255     return Green
256     +++
257     do
258       stringT "orange"
259       return Orange
260
261   ----- Extra parsers -----
262
263   pVIdent :: ReadP Ident
264   pVIdent = do
265     skipSpaces
266     h <- satisfy ('elem' ['A'..'Z'])
267     rest <- munch ('elem' identarr)
268     skipSpaces
269     return $ h:rest
270
271   pSIdent :: ReadP Ident
272   pSIdent = do
273     skipSpaces
274     h <- satisfy ('elem' ['a'..'z'])
275     rest <- munch ('elem' identarr)
276     skipSpaces
277     let ident = h:rest
278     if ident `elem` (reservedWords ++ colourNames)
279     then pfail
280     else return ident
281
282   pInteger :: ReadP Integer
283   pInteger = do
284     skipSpaces
285     n <- munch1 ('elem' ['0'..'9'])
286     skipSpaces
287     -- Note: read is a partial function
288     return (read n :: Integer)
289
290   ----- Helper parsers -----
291
292   bracks :: Char -> ReadP b -> Char -> ReadP b
293   bracks lb a rb = do
294     charT lb
295     b <- a
296     charT rb
297     return b
298
299   stringT :: String -> ReadP ()
300   stringT s = do
301     skipSpaces
302     _ <- string s
303     skipSpaces
304
305   charT :: Char -> ReadP ()
306   charT c = do
307     skipSpaces
308     _ <- char c
309     skipSpaces

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

9.8 Test_Interp.hs

```

1  -----
2  --- Student name: Arni Asgeirsson
3  --- Student KU-id: lwf986
4  -----
5  module Test_Interp
6      (runAllTestsI, runAllTestsIWith)
7      where
8
9  import SalsaAst
10 import Gpx
11 import Test.QuickCheck
12 import qualified Test.QuickCheck as QC
13 import Control.Monad
14 import SalsaInterp
15 import qualified Data.Map as M
16 import Data.Char
17 import Data.List
18 import qualified Data.Maybe as Mb
19 import Test.HUnit
20
21 -----
22 --- Interface to run tests ---
23 -----
24
25 runAllTestsI :: IO ()
26 runAllTestsI = runAllTestsIWith 100
27
28 runAllTestsIWith :: Int -> IO ()
29 runAllTestsIWith n = do
30     putStrLn "----- Running test for the SalsaInterp module -----"
31     putStrLn "-----"
32     putStrLn "----- Running QuickCheck tests -----"
33     putStrLn $ "1. Testing if the interpreter interprets the expected outputs from\n"
34         ++ "    random valid Salsa Programs\n"
35     putStrLn "-----"
36     runQCTest n
37     putStrLn $ "----- Running HUnit tests -----"
38     putStrLn "2. Testing if the Par command works as intended\n"
39     _ <- runTestTT parCases
40     putStrLn "3. Testing if the At command and Group definition works together\n"
41     _ <- runTestTT atgroupCases
42     putStrLn "4. Testing if the Xproj and Yproj expressions work\n"
43     _ <- runTestTT projCases
44     putStrLn "5. Testing if the interpolate function works as intended\n"
45     _ <- runTestTT interpolateCases
46     return () --- Dummy return
47
48 -----
49 ----- QuickCheck Tests -----
50 ----- Test valid input Programs -----
51 -----
52 -----
53 ----- QC test runner -----
54 -----
55 --- TODO allow it to be user defined how many tests it must run
56 runQCTest :: Int -> IO ()
57 runQCTest n = QC.quickCheckWith QC.stdArgs{maxSuccess = n} prop_runProg
58
59

```

```

60 ----- Property -----
61
62 prop_runProg :: TestAnimation -> Bool
63 prop_runProg (TestAnimation ((i,n),o)) = compareAnimations (runProg n i) o
64
65 ----- Test type -----
66
67 newtype TestAnimation = TestAnimation ((Program,Integer), Animation)
68     deriving (Show, Eq)
69
70 instance QC.Arbitrary TestAnimation where
71     arbitrary = do
72         defcoms <- QC.listOf1 $ QC.elements $ definitions++commands
73         (input,output_) <- genManyDefcom ("viewdef":defcoms)
74         return $ TestAnimation (input, output_)
75
76 ----- Definitions -----
77
78 identarr :: String
79 identarr = ['A'..'Z']++['a'..'z']++['0'..'9']++ "_"
80
81 definitions :: [String]
82 definitions = ["viewdef","rectangle", "circle", "view","group"]
83
84 commands :: [String]
85 commands = ["move", "at", "--par",
86             "move","move","move"]
87
88 colours :: [(String,Colour)]
89 colours = [("blue",Blue), ("plum",Plum), ("red",Red),
90            ("green",Green), ("orange",Orange)]
91
92 numbers :: String
93 numbers = ['0'..'9']
94
95 exprList :: [String]
96 exprList = ["const","plus", "minus", "--xproj", "yproj",
97             "const","const","const","const","const"]
98
99 posList :: [String]
100 posList = ["abs", "rel"]
101
102 ----- Generators -----
103
104
105 -- The framerate is intentionally keep low to avoid very very big data sets
106
107 genManyDefcom :: [String] -> QC.Gen (([DefCom],Integer),Animation)
108 genManyDefcom [] = error "Cannot parse an empty list of definitions or commands"
109 genManyDefcom words_ = do
110     n <- QC.elements ['1'..'9']
111     let framerate = read [n]::Integer
112         init_ = (createEmptyContext framerate,[],([],[]))
113     (_,all_defcoms,all_anim) <- foldM f init_ words_
114     return ((all_defcoms,framerate),all_anim)
115     where
116         f (acontext,acc,anim) word = do
117             (context,defcoms,new_anim) <- genDefcom word acontext anim
118             if word `elem` commands && defcoms /= []
119             then
120                 let (view,frames) = new_anim
121                     next = frames++[]

```

```

122         in
123         return (context , acci++defcoms ,(view ,next))
124     else
125         return (context , acci++defcoms ,new_anim)
126 genDefcom :: String -> Context -> Animation -> QC.Gen (Context,[DefCom],Animation)
127 genDefcom "viewdef" c@(Context (env,_,n) state) a@(views,frames) = do
128     vident <- genVident
129     if isInEnvironment vident env
130     then
131         return (c,[],a)
132     else do
133         expw_ <- genExpr env
134         expw_ <- genExpr env
135         expw_ <- forcePositive expw_ state
136         expw_ <- forcePositive expw_ state
137         w <- evalExprQC expw state
138         h <- evalExprQC expw state
139         let def = Viewdef vident expw expw
140             env' = M.insert vident def env
141         return (Context (env',[vident],n) state,[Def def], (views++[(vident,w,h)],frames))
142 genDefcom "rectangle" c@(Context (env,active,n) state) a@(views,frames) = do
143     sident <- genSident
144     if isInEnvironment sident env
145     then
146         return (c,[],a)
147     else do
148         expx <- genExpr env
149         expy <- genExpr env
150         expw_ <- genExpr env
151         expw_ <- forcePositive expw_ state
152         expw_ <- forcePositive expw_ state
153         expw_ <- forcePositive expw_ state
154         x <- evalExprQC expx state
155         y <- evalExprQC expy state
156         w <- evalExprQC expw state
157         h <- evalExprQC expw state
158         (col,col_type) <- genColour
159         let (rest,[last_]) = getLast frames
160             instrs = map (\viewName -> DrawRect x y w h viewName col) active
161             def = Rectangle sident expx expy expw expw expw col_type
162             env' = M.insert sident def env
163             list = map (\viewName -> (viewName,(x,y))) active
164             state' = M.insert sident list state
165         return (Context (env',active,n) state',[Def def], (views,rest++[last_++instrs]))
166 genDefcom "circle" c@(Context (env,active,n) state) a@(views,frames) = do
167     sident <- genSident
168     if isInEnvironment sident env
169     then
170         return (c,[],a)
171     else do
172         expx <- genExpr env
173         expy <- genExpr env
174         expr_ <- genExpr env
175         expr_ <- forcePositive expr_ state
176         x <- evalExprQC expx state
177         y <- evalExprQC expy state
178         r <- evalExprQC expr state
179         (col,col_type) <- genColour
180         let (rest,[last_]) = getLast frames
181             instrs = map (\viewName -> DrawCirc x y r viewName col) active
182             def = Circle sident expx expy expr col_type
183             env' = M.insert sident def env

```

Assignment Project Exam Help
<https://powcoder.com>
 Add WeChat powcoder

```

184         list = map (\viewName -> (viewName,(x,y))) active
185         state' = M.insert sident list state
186         return (Context (env',active,n) state',[Def def],(views,rest++[last_++instrs]))
187 genDefcom "view" c@(Context (env,_,n) state) a@anim =
188     let list = M.toList env
189         flist = filter (\(_,def) -> case def of
190             (Group _) -> True
191             (View _) -> True
192             _ -> False
193             ) list
194     in
195     if null flist
196     then return (c,[], a)
197     else do
198         (id_,some_def) <- QC.elements flist
199         let new_active = case some_def of
200             (Group _ g) -> g
201             (View v) -> [v]
202             _ -> error "Shouldn't be possible due to filtering above"
203         return (Context (env,new_active,n) state,[Def $ View id_],anim)
204 genDefcom "group" c@(Context (_,[],_) _) a =
205     return (c,[], a)
206 genDefcom "group" c@(Context (env,active,n) state) a = do
207     vident <- genVident
208     if isInEnvironment vident env
209     then
210         return (c,[], a)
211     else do
212         new_actives <- QC.listOf1 $ QC.elements active
213         let new_actives = removeDuplex new_actives_
214             def = Group vident new_actives
215             env' = M.insert vident def env
216         return (Context (env',new_actives,n) state,[Def def],a)
217 genDefcom "move" c@(Context (env,active,n) state) a@(views,frames) =
218     let list = M.toList env
219         flist = filter (\(x:_,_) -> isLower x) list
220     in
221     if null flist
222     then return (c,[], a)
223     else do
224         ids <- QC.listOf1 $ QC.elements flist
225         let ids2 = removeDuplex ids
226             expPos <- genPos env
227             (all_instr,new_state) <- foldM (f_ active expPos n state) ([],state) ids2
228             let (rest,[last_]) = getLast frames
229                 next1 = rest++[last_++all_instr]
230                 ids_ = map fst ids2
231                 def = Move ids_ expPos
232             return (Context (env,active,n) new_state,[Com def],(views,next1))
233 genDefcom "at" c@(Context (env,active,n) state) a =
234     let list = M.toList env
235         flist = filter (\(_,def) -> case def of
236             (Group _) -> True -- Uncomment to see doom and destruction!
237             (View _) -> True
238             _ -> False
239             ) list
240     in
241     if null flist
242     then return (c,[], a)
243     else do
244         (id_,some_def) <- QC.elements flist
245         let tmp_active = case some_def of

```

```

246         (Group _ g) -> g
247         (View v) -> [v]
248         _ -> error "Shouldn't be possible due to the filtering above"
249     middle_cmd <- QC.elements commands
250     middle <- genDefcom middle_cmd (Context (env,tmp_active,n) state) a
251     let (Context (env',_,_) state', a_com, anim') = middle
252     case a_com of
253     [] ->
254         return (c,[],a)
255     [Com some] ->
256         return (Context (env',active,n) state', [Com $ At some id_], anim')
257     _ -> error "Shouldn't be possible"
258 genDefcom "par" c a = do
259     cmd1 <- QC.elements commands
260     cmd2 <- QC.elements commands
261     res1 <- genDefcom cmd1 c a
262     let (con1, defcom1, anim1) = res1
263     res2 <- genDefcom cmd2 con1 anim1
264     let (con2, defcom2, anim2) = res2
265     return (case (defcom1,defcom2) of
266         ([],_) ->
267             (c, [], a)
268         (_,[]) ->
269             (c, [], a)
270         ([Com com1],[Com com2]) ->
271             (con2, [Com $ Par com1 com2], anim2)
272         _ -> error "Shouldn't be possible")
273 genDefcom "error" _ _ = error "Can not pass 'error' into a Defcom"
274
275 forcePositive :: Expr -> State_ -> Gen Expr
276 forcePositive exp_ s = do
277     val <- evalExprQC exp_ s
278     return (if val < 0
279         then
280             exp_
281         else
282             Plus exp_ (Const (val*(-2))))
283
284 -- COM assumes that it is in state
285 -- TODO RENAME
286 f_ :: Ord t => [ViewName] -> Pos -> Integer -> State_ ->
287     ([GpxInstr], M.Map t [(ViewName, (Integer, Integer))]) ->
288     (t, Definition) -> Gen ([GpxInstr], M.Map t [(ViewName, (Integer, Integer))])
289 f_ active pos n s acc (id_,def) =
290     foldM (\(acc_instrs, acc_state) a ->
291         let positions = lookupKey id_ acc_state
292         in
293             case lookup a positions of
294             Just old_pos -> do
295                 next_pos <- evalPos_ old_pos pos s
296                 let next_state = M.insert id_ (map (\(vn,p) -> if vn == a
297                     then (vn,next_pos)
298                     else (vn,p)
299                     ) positions) acc_state
300                 instr <- genInstrs old_pos next_pos def n a s
301                 return (acc_instrs++instr,next_state)
302             Nothing ->
303                 return (acc_instrs,acc_state)) acc active
304
305 genInstrs :: (Integer, Integer)
306           -> (Integer, Integer)
307           -> Definition

```

```

308         -> Integer
309         -> ViewName
310         -> State
311         -> Gen [GpxInstr]
312 genInstrs opos npos (Rectangle _ _ _ ew eh ecol) n view s = do
313   w <- evalExprQC ew s
314   h <- evalExprQC eh s
315   col <- evalColourQc ecol
316   let positions = interpolate n opos npos
317   return $ map (\(x,y) -> DrawRect x y w h view col) positions
318 genInstrs opos npos (Circle _ _ _ er ecol) n view s = do
319   r <- evalExprQC er s
320   col <- evalColourQc ecol
321   let positions = interpolate n opos npos
322   return $ map (\(x,y) -> DrawCirc x y r view col) positions
323 genInstrs _ _ _ _ _ = error "Cannot generate instructions if not given a shape"
324
325 evalPos_ :: (Integer, Integer)
326           -> Pos -> State_ -> Gen (Integer, Integer)
327 evalPos_ _ (Abs expx expy) s = do
328   x2 <- evalExprQC expx s
329   y2 <- evalExprQC expy s
330   return (x2,y2)
331 evalPos_ (x,y) (Rel expx expy) s = do
332   x2 <- evalExprQC expx s
333   y2 <- evalExprQC expy s
334   return (x+x2,y+y2)
335
336 genPos :: Environment -> QC.Gen Pos
337 genPos env = do
338   pos <- QC.elements posList
339   _genPos pos env
340
341 _genPos :: String -> Environment -> QC.Gen Pos
342 _genPos "abs" env = do
343   expx <- genExpr env
344   expy <- genExpr env
345   return (Abs expx expy)
346 _genPos "rel" env = do
347   expx <- genExpr env
348   expy <- genExpr env
349   return (Rel expx expy)
350 _genPos s _ = error $ "Cannot parse "++s++" into an Pos"
351
352 evalColourQc :: Colour -> QC.Gen String
353 evalColourQc Blue = return "blue"
354 evalColourQc Plum = return "plum"
355 evalColourQc Red = return "red"
356 evalColourQc Green = return "green"
357 evalColourQc Orange = return "orange"
358
359 evalExprQC :: Expr -> State_ -> QC.Gen Integer
360 evalExprQC (Const n) _ = return n
361 evalExprQC (Plus e1 e2) s = do
362   n1 <- evalExprQC e1 s
363   n2 <- evalExprQC e2 s
364   return $ n1 + n2
365 evalExprQC (Minus e1 e2) s = do
366   n1 <- evalExprQC e1 s
367   n2 <- evalExprQC e2 s
368   return $ n1 - n2
369 evalExprQC (Xproj id_) s = do

```

```

370 let max_i = toInteger(maxBound :: Int)
371 (x,_) = foldl getLowestPosition (max_i,max_i) $ lookupKey id_ s
372 return x
373 evalExprQC (Yproj id_) s = do
374 let max_i = toInteger(maxBound :: Int)
375 (x,y) = foldl getLowestPosition (max_i,max_i) $ lookupKey id_ s
376 return y
377
378 genExpr :: Environment -> QC.Gen Expr
379 genExpr env = do
380   expr <- QC.elements exprList
381   _genExpr expr env
382
383 -- If I have to create a xproj or yproj, but no shape definition
384 -- has been made yet, a Const will be returned instead.
385 _genExpr :: String -> Environment -> QC.Gen Expr
386 _genExpr "plus" env = do
387   (exp1) <- genExpr env
388   (exp2) <- genExpr env
389   return (Plus exp1 exp2)
390 _genExpr "minus" env = do
391   (exp1) <- genExpr env
392   (exp2) <- genExpr env
393   return (Minus exp1 exp2)
394 _genExpr "const" _ = do
395   n <- genNumber
396   return (Const (read n :: Integer))
397 _genExpr "xproj" env = projHelper xproj env
398 _genExpr "yproj" env = projHelper yproj env
399 _genExpr s _ = error $ "Cannot parse "++s++" into an Expr"
400
401 projHelper :: (String -> Expr) -> Environment -> QC.Gen Expr
402 projHelper a env = do
403   let list = M.toList env
404       flist = filter (\(x:_,_) => isLower x) list
405   in
406   if null flist
407   then
408     _genExpr "const" env
409   else do
410     (sid,_) <- QC.elements flist
411     return $ a sid
412
413 genColour :: QC.Gen (String,Colour)
414 genColour = QC.elements colours
415
416 genVident :: QC.Gen String
417 genVident = do
418   h <- QC.elements ['A'..'Z']
419   rest <- QC.listOf $ QC.elements identarr
420   return $ h:rest
421
422 genSident :: QC.Gen String
423 genSident = do
424   h <- QC.elements ['a'..'z']
425   rest <- QC.listOf $ QC.elements identarr
426   return $ h:rest
427
428 genNumber :: QC.Gen String
429 genNumber = QC.listOf1 $ QC.elements numbers
430
431

```



```

432 ----- Helper Functions -----
433 -----
434
435 compareAnimations :: Animation -> Animation -> Bool
436 compareAnimations (views1,frames1) (views2,frames2) =
437   let (b1,_) = g views1 views2
438       b2 = f frames1 frames2
439       f [] [] = True
440       f [] _ = False
441       f _ [] = False
442       f (x:xs) (y:ys) = let (b,_) = g x y
443                           in
444                             b && f xs ys
445       g x1 x2 = foldl1 (\(a,v2) v -> if a && v 'elem' v2
446                               then (True,delete v v2)
447                               else (False,[]))
448                     (True,x2) x1
449   in
450     b1 && b2
451
452 isInEnvironment :: Ord k => k -> M.Map k a -> Bool
453 isInEnvironment id_ env = case M.lookup id_ env of
454   Just _ -> True
455   Nothing -> False
456
457 removeDuplex :: Eq a => [a] -> [a]
458 removeDuplex [] = []
459 removeDuplex (x:xs) = if x 'elem' xs
460                       then removeDuplex xs
461                       else x:removeDuplex xs
462
463 ----- Copied from the SalsaInterp.hs file -----
464
465 data Context = Context ConEnvironment State_
466               deriving (Show)
467
468 type ConEnvironment = (Environment, [Ident], Integer)
469 type Environment = M.Map Ident Definition
470 type State_ = M.Map Ident [(Ident, Position)]
471
472
473 createEmptyContext :: Integer -> Context
474 createEmptyContext n = Context (M.empty, [], n) M.empty
475
476 lookupKey :: Ord a => a -> M.Map a b -> b
477 lookupKey key env =
478   Mb.fromMaybe (error "Tried to look up unknown key ")
479   (M.lookup key env)
480
481 getLast :: [a] -> ([a],[a])
482 getLast [] = ([],[])
483 getLast (x:[]) = ([],[x])
484 getLast (x:xs) = let (rest,last_) = getLast xs
485                   in
486                     (x:rest,last_)
487
488 getLowestPosition :: Position -> (ViewName,Position) -> Position
489 getLowestPosition (l_x,l_y) (_,(x,y)) = let
490   n_x = if x < l_x then x else l_x
491   n_y = if y < l_y then y else l_y
492   in
493     (n_x,n_y)

```

```

494
495
496 ----- HUnit tests -----
497 ----- Par tests -----
498 -----
499
500
501 parCases :: Test
502 parCases = TestLabel "Test cases for the Par command"
503             $ TestList [testP1]
504
505 -- Test that par command works in a simple example
506 testP1 :: Test
507 testP1 = let s = [ Def (Viewdef "A" (Const 200) (Const 12))
508                   , Def (Circle "b" (Const 10) (Const 10) (Const 5) Green)
509                   , Def (Circle "c" (Const 10) (Const 10) (Const 5) Blue)
510                   , Com (Par (Move ["b"] (Abs (Const 20) (Const 50))) (Move ["c"] (Abs (Const 100) (Const 50))))
511                   a = ([("A",200,12)],[[DrawCirc 10 10 5 "A" "green",DrawCirc 10 10 5 "A" "blue"
512                                           ,DrawCirc 12 18 5 "A" "green",DrawCirc 14 26 5 "A" "green"
513                                           ,DrawCirc 16 34 5 "A" "green",DrawCirc 18 42 5 "A" "green"
514                                           ,DrawCirc 20 50 5 "A" "green",DrawCirc 28 18 5 "A" "blue"
515                                           ,DrawCirc 46 26 5 "A" "blue",DrawCirc 64 34 5 "A" "blue"
516                                           ,DrawCirc 82 42 5 "A" "blue",DrawCirc 100 50 5 "A" "blue"],[]])
517                   in TestCase $ assertEquals "a" (runProg 5 s)
518
519
520 -----
521 ----- At/Group tests -----
522 -----
523
524
525 atgroupCases :: Test
526 atgroupCases = TestLabel "Test cases for testing At and Group"
527                 $ TestList [testAg1]
528
529 -- Test that the At command and Group definition works together
530
531 testAg1 :: Test
532 testAg1 = let s = [ Def (Viewdef "A" (Const 200) (Const 12))
533                   , Def (Circle "b" (Const 200) (Const 20) (Const 5) Green)
534                   , Def (Viewdef "B" (Const 10) (Const 500))
535                   , Def (Group "C" ["A","B"])
536                   , Com (At (Move ["b"] (Rel (Const 20) (Const 50))) "C")]
537                   a = ([("A",200,12),("B",10,500)],
538                       [[DrawCirc 200 20 5 "A" "green"
539                           ,DrawCirc 210 45 5 "A" "green",DrawCirc 220 70 5 "A" "green"],[]])
540                   in TestCase $ assertEquals "a" (runProg 2 s)
541
542 -----
543 ----- Proj tests -----
544 -----
545
546 projCases :: Test
547 projCases = TestLabel "Test cases for the [XY]proj expressions"
548                 $ TestList [testPr1]
549
550 -- Test that Xproj and Yproj works in a simple example
551 testPr1 :: Test
552 testPr1 = let s = [ Def (Viewdef "A" (Const 200) (Const 12))
553                   , Def (Circle "b" (Const 200) (Const 20) (Const 5) Green)
554                   , Def (Circle "c" (Const 10) (Const 100) (Const 5) Blue)
555                   , Com (Move ["b"] (Abs (Yproj "c") (Xproj "b")))]

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

556         a = ([("A",200,12)],[[DrawCirc 200 20 5 "A" "green",DrawCirc 10 100 5 "A" "blue"
557                               ,DrawCirc 150 110 5 "A" "green",DrawCirc 100 200 5 "A" "green"]],
558         in TestCase $ assertEquals "a" a (runProg 2 s)
559
560 -----
561 interpolate tests -----
562 -----
563
564 interpolateCases :: Test
565 interpolateCases = TestLabel "Test cases for the interpolate function"
566                   $ TestList [testIp1, testIp2, testIp3, testIp4, testIp5]
567
568 -- Test with 0 frame rate
569 testIp1 :: Test
570 testIp1 = let s = interpolate 0 (0,0) (10,10)
571           a = []
572           d = "0 frame rate should return []"
573           in TestCase $ assertEquals d a s
574
575 -- Test with 1 frame rate
576 testIp2 :: Test
577 testIp2 = let s = interpolate 1 (0,0) (10,10)
578           a = [(10,10)]
579           d = "1 frame rate should return the end point"
580           in TestCase $ assertEquals d a s
581
582 -- Test with 5 frame rate
583 testIp3 :: Test
584 testIp3 = let s = interpolate 5 (0,0) (10,10)
585           a = [(2,2),(4,4),(6,6),(8,8),(10,10)]
586           d = "5 frame rate should return a list of 5 points"
587           in TestCase $ assertEquals d a s
588
589 -- Test with 5 and negative direction
590 testIp4 :: Test
591 testIp4 = let s = interpolate 5 (10,10) (0,0)
592           a = [(8,8),(6,6),(4,4),(2,2),(0,0)]
593           d = "5 frame rate should return a list of 5 points"
594           in TestCase $ assertEquals d a s
595
596
597 -- Test with same points
598 testIp5 :: Test
599 testIp5 = let s = interpolate 3 (5,5) (5,5)
600           a = [(5,5),(5,5),(5,5)]
601           d = "interpolate from point a to a should only a"
602           in TestCase $ assertEquals d a s

```

9.9 Test_Parser.hs

```

1 -----
2 -- Student name: Arni Asgeirsson
3 -- Student KU-id: lwf986
4 -----
5 module Test_Parser
6     (runAllTests, runAllTestsWith)
7     where
8
9 import Test.HUnit
10 import SalsaAst

```

```

11 import SalsaParser
12 import Test.QuickCheck
13 import qualified Test.QuickCheck as QC
14 import Control.Monad
15 import Control.Exception
16
17 -----
18 ----- Interface to run tests -----
19 -----
20
21 runAllTests :: IO Bool
22 runAllTests = runAllTestsWith 100
23
24 runAllTestsWith :: Int -> IO Bool
25 runAllTestsWith n = do
26   putStrLn "-----"
27   putStrLn "----- Running tests for the SalsaParser module -----"
28   putStrLn "-----\n"
29   putStrLn "----- Running QuickCheck tests -----"
30   putStrLn $ "1. Testing if the parser parses the expected outputs from random"
31   ++ " valid input strings"
32   putStrLn "Might take a few seconds ...\n"
33   runQCTest n
34   putStrLn "\n----- Running HUnit tests -----"
35   putStrLn "2. Testing if the precedence and associativity works as intended\n"
36   _ <- runTestTT precedenceCases
37   putStrLn $ "\n3. Testing if the expected errors occur with specific invalid"
38   ++ " input strings\n"
39   _ <- runTestTT errorCases
40   putStrLn "\n----- Running Unit tests -----"
41   putStrLn "4. Testing if an empty file parses as expected\n"
42   b1 <- testF1
43   print b1
44   putStrLn "\n5. Testing if a simple Salsa file parses as expected\n"
45   b2 <- testF2
46   print b2
47   putStrLn "\n6. Testing if a more advanced Salsa file parses as expected\n"
48   b3 <- testF3
49   print b3
50   putStrLn "\n7. Testing if an invalid Salsa program from file parses as expected\n"
51   b4 <- testF4
52   print b4
53   putStrLn "\n8. Testing if a non-existing file raises the expected error\n"
54   testF5
55
56 -----
57 ----- QuickCheck Tests -----
58 ----- Test valid input strings -----
59 -----
60
61 ----- Definitions -----
62
63 identarr :: String
64 identarr = ['A'..'Z']++['a'..'z']++['0'..'9']++["_"]
65
66 definitions :: [String]
67 definitions = ["viewdef", "rectangle", "circle", "view", "group"]
68
69 commands :: [String]
70 commands = ["move", "at", "par",
71            "move", "move", "move"]
72

```

```

73 colours :: [(String, Colour)]
74 colours = [("blue", Blue), ("plum", Plum), ("red", Red),
75           ("green", Green), ("orange", Orange)]
76
77 whiteSpaces :: [String]
78 whiteSpaces = [" ", "\n"]
79
80 numbers :: String
81 numbers = ['0'..'9']
82
83 exprList :: [String]
84 exprList = ["plus", "minus", "const", "xproj", "yproj",
85           "const", "const", "const", "const"]
86
87 posList :: [String]
88 posList = ["abs", "rel"]
89
90 ----- Test type -----
91
92 newtype TestProgram = TestProgram (String, Either Error Program)
93                       deriving (Show, Eq)
94
95 instance QC.Arbitrary TestProgram where
96   arbitrary = do
97     defcoms <- QC.listOf1 $ QC.elements $ definitions++commands
98     (input,output_) <- genManyDefcom defcoms
99     return $ TestProgram (input, Right output_)
100 ----- Property -----
101
102 prop_pProgram :: TestProgram -> Bool
103 prop_pProgram (TestProgram (i,o)) = parseString i == o
104 ----- QC test runner -----
105
106 runQCTest :: Int -> IO ()
107 runQCTest n = QC.quickCheckWith QC.stdArgs{maxSuccess = n } prop_pProgram
108 ----- Generator -----
109
110 genManyDefcom :: [String] -> QC.Gen (String, [DefCom])
111 genManyDefcom [] = error "Cannot parse an empty list of definitions or commands"
112 genManyDefcom words_ = do
113   result <- mapM genDefcom words_
114   foldM f ("",[[]]) result
115   where
116     f (acci,acco) (i,o) = return (acci++i, acco++o)
117
118 genDefcom :: String -> QC.Gen (String, [DefCom])
119 genDefcom "viewdef" = do
120   vident <- genVident
121   (w,expw) <- genExpr
122   (h,exph) <- genExpr
123   input <- insertWhiteSpaces1 ["viewdef",vident,w,h]
124   return (input, [Def $ Viewdef vident expw exph])
125 genDefcom "rectangle" = do
126   sident <- genSident
127   (x,expx) <- genExpr
128   (y,expy) <- genExpr
129   (w,expw) <- genExpr
130   (h,exph) <- genExpr
131   (col,col_type) <- genColour

```

```

135 input <- insertWhiteSpaces1 ["rectangle",sident,x,y,w,h,col]
136 return (input, [Def $ Rectangle sident expx expy expw expw expw col_type])
137 genDefcom "circle" = do
138   sident <- genSident
139   (x,expx) <- genExpr
140   (y,expy) <- genExpr
141   (r,expr) <- genExpr
142   (col,col_type) <- genColour
143   input <- insertWhiteSpaces1 ["circle",sident,x,y,r,col]
144   return (input, [Def $ Circle sident expx expy expr col_type])
145 genDefcom "view" = do
146   vident <- genVident
147   input <- insertWhiteSpaces1 ["view",vident]
148   return (input, [Def $ View vident])
149 genDefcom "group" = do
150   vident <- genVident
151   vidents <- QC.listOf1 genVident
152   input <- insertWhiteSpaces1 $ ["group",vident,"["++vidents++"]"]
153   return (input, [Def $ Group vident vidents])
154 genDefcom "move" = do
155   sidents <- QC.listOf1 genSident
156   (pos,expPos) <- genPos
157   input <- insertWhiteSpaces1 $ [{""]++sidents++["->",pos,"}"]
158   return (input, [Com $ Move sidents expPos])
159 genDefcom "at" = do
160   word <- QC.elements commands
161   (cmd,Com cmdexp:[[]]) <- genDefcom word
162   vident <- genVident
163   input <- insertWhiteSpaces1 [{"{",cmd,"@",vident,"}"}]
164   return (input, [Com $ At cmdexp vident])
165 genDefcom "par" = do
166   word1 <- QC.elements commands
167   word2 <- QC.elements commands
168   (cmd1,Com cmdexp1:[[]]) <- genDefcom word1
169   (cmd2,Com cmdexp2:[[]]) <- genDefcom word2
170   input <- insertWhiteSpaces1 [{"{",cmd1,"||",cmd2,"}"}]
171   return (input, [Com $ Par cmdexp1 cmdexp2])
172 genDefcom s = error $ "Cannot parse '"++s++"' into a DefCom"
173
174 genPos :: QC.Gen (String, Pos)
175 genPos = do
176   pos <- QC.elements posList
177   _genPos pos
178
179 _genPos :: String -> QC.Gen (String, Pos)
180 _genPos "abs" = do
181   (x,expx) <- genExpr
182   (y,expy) <- genExpr
183   input <- insertWhiteSpaces1 ["(",x,"-",y,")"]
184   return (input, Abs expx expy)
185 _genPos "rel" = do
186   (x,expx) <- genExpr
187   (y,expy) <- genExpr
188   input <- insertWhiteSpaces1 ["+", "(" ,x,"-",y,")"]
189   return (input, Rel expx expy)
190 _genPos s = error $ "Cannot parse '"++s++"' into a Pos"
191
192 genExpr :: QC.Gen (String, Expr)
193 genExpr = do
194   expr <- QC.elements exprList
195   _genExpr expr
196

```

```

197 _genExpr :: String -> QC.Gen (String, Expr)
198 _genExpr "plus" = do
199   (e1, exp1) <- genExpr
200   (e2, exp2) <- genExpr
201   input <- insertWhiteSpaces ["(", e1, "+", e2, ")"]
202   return (input, Plus exp1 exp2)
203 _genExpr "minus" = do
204   (e1, exp1) <- genExpr
205   (e2, exp2) <- genExpr
206   input <- insertWhiteSpaces ["(", e1, "-", e2, ")"]
207   return (input, Minus exp1 exp2)
208 _genExpr "const" = do
209   n <- genNumber
210   input <- insertWhiteSpaces ["(", n, ")"]
211   return (input, Const (read n :: Integer))
212 _genExpr "xproj" = do
213   sident <- genSident
214   input <- insertWhiteSpaces ["(", sident, ".", "x", ")"]
215   return (input, Xproj sident)
216 _genExpr "yproj" = do
217   sident <- genSident
218   input <- insertWhiteSpaces ["(", sident, ".", "y", ")"]
219   return (input, Yproj sident)
220 _genExpr s = error $ "Cannot parse " ++ s ++ " into an Expr"
221
222 genColour :: QC.Gen (String, Colour)
223 genColour = QC.elements colours
224
225 genVident :: QC.Gen String
226 genVident = do
227   h <- QC.elements ['A'.. 'Z']
228   rest <- QC.listOf $ QC.elements identarr
229   return $ h:rest
230
231 genSident :: QC.Gen String
232 genSident = do
233   h <- QC.elements ['a'.. 'z']
234   rest <- QC.listOf $ QC.elements identarr
235   return $ h:rest
236
237 genNumber :: QC.Gen String
238 genNumber = QC.listOf1 $ QC.elements numbers
239
240 insertWhiteSpaces :: [String] -> QC.Gen String
241 insertWhiteSpaces = wsHelper QC.listOf
242
243 insertWhiteSpaces1 :: [String] -> QC.Gen String
244 insertWhiteSpaces1 = wsHelper QC.listOf1
245
246 wsHelper :: (QC.Gen String -> QC.Gen [String]) -> [String] -> QC.Gen String
247 wsHelper m words_ = do
248   initWhite <- m $ QC.elements whiteSpaces
249   foldM f (concat initWhite) words_
250   where
251     f acc word = do
252       whiteSpaces <- QC.listOf1 $ QC.elements whiteSpaces
253       return $ acc ++ word ++ concat whiteSpaces
254
255 -----
256 ----- HUnit tests -----
257 ----- Precedence tests -----
258 -----

```

Assignment Project Exam Help
<https://powcoder.com>
 Add WeChat powcoder

```

259 precedenceCases :: Test
260 precedenceCases = TestLabel "Test cases for precedence"
261   $ TestList [testP1, testP2, testP3, testP4, testP5,
262               testP6, testP7, testP8, testP9, testP10,
263               testP11]
264
265 -- Show that @ and || are both left associative
266 testP1 :: Test
267 testP1 = let s = "a -> (0,0) @ A @ B @ C"
268           a = "{{{ a -> (0,0) @ A } @ B } @ C }"
269           d = "@ must be left associative"
270           in TestCase $ assertEquals d (parseString a) (parseString s)
271
272 testP2 :: Test
273 testP2 = let s = "a -> (0,0) || b->(0,0) || c->(0,0)"
274           a = "{{{a -> (0,0)} || b->(0,0)} || c->(0,0)}"
275           d = "|| must be left associative"
276           in TestCase $ assertEquals d (parseString a) (parseString s)
277
278 -- Left associativity of + and -
279 testP3 :: Test
280 testP3 = let s = "viewdef A 1 1+5+2"
281           a = "viewdef A 1 (1+5)+2"
282           d = "+ must be left associative"
283           in TestCase $ assertEquals d (parseString a) (parseString s)
284
285 testP4 :: Test
286 testP4 = let s = "viewdef A 1 1-5-2"
287           a = "viewdef A 1 (1-5)-2"
288           d = "- must be left associative"
289           in TestCase $ assertEquals d (parseString a) (parseString s)
290
291 testP5 :: Test
292 testP5 = let s = "viewdef A 1 1+5-2-5"
293           a = "viewdef A 1 (1+5)-2-5"
294           d = "+ & - must be left associative"
295           in TestCase $ assertEquals d (parseString a) (parseString s)
296
297 -- Show that @ has higher precedence than ||
298 testP6 :: Test
299 testP6 = let s = "a -> (0,0) || {b->(0,0) @ A}"
300           a = "a -> (0,0) || {b->(0,0) @ A}"
301           d = "@ should have higher precedence than ||"
302           in TestCase $ assertEquals d (parseString a) (parseString s)
303
304 testP7 :: Test
305 testP7 = let s = "a -> (0,0) || b->(0,0) @ A || c->(0,0) @ B @ C"
306           a = "a -> (0,0) || {b->(0,0) @ A} || {{c->(0,0) @ B} @ C}"
307           d = "@ should have higher precedence than ||"
308           in TestCase $ assertEquals d (parseString a) (parseString s)
309
310 -- Show that + and - has the same precedence (ie it maintains its order)
311 testP8 :: Test
312 testP8 = let s = "viewdef A 1 1+5-4"
313           a = "viewdef A 1 ((1+5)-4)"
314           d = "+ & - should have the same precedence"
315           in TestCase $ assertEquals d (parseString a) (parseString s)
316
317 testP9 :: Test
318 testP9 = let s = "viewdef A 1 1-5+4"
319           a = "viewdef A 1 ((1-5)+4)"
320           d = "+ & - should have the same precedence"
321           in TestCase $ assertEquals d (parseString a) (parseString s)
322
323 -- Show that . has higher precedence than + and -
324 testP10 :: Test

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder


```

321 testP10 = let s = "viewdef A 1 r.x + c.y"
322           a = "viewdef A 1 (r.x) + (c.y)"
323           d = ". should have higher precedence than +"
324           in TestCase $ assertEquals d (parseString a) (parseString s)
325 testP11 :: Test
326 testP11 = let s = "viewdef A 1 r.x - c.y"
327           a = "viewdef A 1 (r.x) - (c.y)"
328           d = ". should have higher precedence than -"
329           in TestCase $ assertEquals d (parseString a) (parseString s)
330
331 -----
332 ----- Error tests -----
333 -----
334 errorCases :: Test
335 errorCases = TestLabel "Test cases for invalid input strings"
336             $ TestList [testE1, testE2, testE3, testE4, testE5,
337                       testE6, testE7, testE8, testE9, testE10,
338                       testE11, testE12, testE13, testE14, testE15,
339                       testE16, testE17, testE18, testE19, testE20,
340                       testE21, testE22, testE23, testE24, testE25,
341                       testE26, testE27, testE28, testE29, testE30,
342                       testE31, testE32, testE33, testE34, testE35,
343                       testE36, testE37, testE38, testE39, testE40,
344                       testE41]
345
346 -- Parse empty string
347 testE1 :: Test
348 testE1 = let s = ""
349         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
350
351 -- Using sident in place of vident
352 testE2 :: Test
353 testE2 = let s = "triangle sident 4 5"
354         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
355
356 testE3 :: Test
357 testE3 = let s = "view sident"
358         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
359
360 testE4 :: Test
361 testE4 = let s = "group sident [A,B,C]"
362         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
363
364 testE5 :: Test
365 testE5 = let s = "group Vident [A,B,sident]"
366         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
367
368 -- Using vident in place of sident
369 testE7 :: Test
370 testE7 = let s = "rectangle Vident 1 2 3 4 green"
371         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
372
373 testE8 :: Test
374 testE8 = let s = "circle Vident 1 2 3 red"
375         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
376
377 testE9 :: Test
378 testE9 = let s = "{ Vident -> (1,1) } @ A"
379         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
380
381 testE10 :: Test
382 testE10 = let s = "{ a b c Vident -> (1,1) } @ A"
383         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
384
385 testE11 :: Test
386 testE11 = let s = "viewdef A 1 (1 + Vident . x)"

```

```

383         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
384
385 -- Using wrong types of characters
386 testE12 :: Test
387 testE12 = let s = "viewdef _A 1 2"
388         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
389 testE13 :: Test
390 testE13 = let s = "viewdef A+a 1 2"
391         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
392 testE14 :: Test
393 testE14 = let s = "view Bæ"
394         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
395 testE15 :: Test
396 testE15 = let s = "circle a 1 2 (a . z) red"
397         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
398
399 -- Using numbers instead of letters, (same as using wrong characters?)
400 testE16 :: Test
401 testE16 = let s = "5 -> (0,0)"
402         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
403 testE17 :: Test
404 testE17 = let s = "group 2 [ A ]"
405         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
406
407 -- Using letters instead of numbers
408 testE18 :: Test
409 testE18 = let s = "viewdef A b 4"
410         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
411 testE19 :: Test
412 testE19 = let s = "viewdef A 4 (a+2)"
413         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
414
415 -- TODO does it make sense to test for this?
416 -- Using wrong parent tests
417 testE20 :: Test
418 testE20 = let s = "( a -> (0,0) )"
419         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
420 testE21 :: Test
421 testE21 = let s = "group A { 1 }"
422         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
423 testE22 :: Test
424 testE22 = let s = "{ a -> [0,0] }"
425         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
426
427 -- TODO does it make sense to test for this?
428 -- Using non-existing operator
429 testE23 :: Test
430 testE23 = let s = "a -> - (1,2)"
431         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
432 testE24 :: Test
433 testE24 = let s = "viewdef A 1 (2 * 4)"
434         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
435 testE25 :: Test
436 testE25 = let s = "circle a 3 1 ( a , x) red"
437         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
438
439 -- Using a reserved word as sident (res+colour)
440 testE26 :: Test
441 testE26 = let s = "rectangle viewdef 1 2 3 4 green"
442         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
443 testE27 :: Test
444 testE27 = let s = "rectangle rectangle 1 2 3 4 green"

```

```

445         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
446 testE28 :: Test
447 testE28 = let s = "circle a 1 2 (1 + group . x)"
448         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
449 testE29 :: Test
450 testE29 = let s = "green -> (6,66)"
451         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
452
453 -- TODO does it make sense to test for this?
454 -- Using invalid colour name
455 testE30 :: Test
456 testE30 = let s = "circle a 1 2 3 purple"
457         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
458
459 -- Wrong whitespace inbetween stuff
460 testE31 :: Test
461 testE31 = let s = "circle a 3l ( a , x) red"
462         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
463 testE32 :: Test
464 testE32 = let s = "circlea 3 1 ( a , x) red"
465         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
466 testE33 :: Test
467 testE33 = let s = "viewdefA 1 2"
468         in TestCase $ assertEquals ""
469             (Right [Def $ Viewdef "A" (Const 1) (Const 2)])
470             (parseString s)
471 testE34 :: Test
472 testE34 = let s = "group A [30]"
473         in TestCase $ assertEquals ""
474             (Right [Def $ Group "A" ["BCD"]])
475             (parseString s)
476
477 -- Grouping zero ident
478 testE35 :: Test
479 testE35 = let s = "group A [ ]"
480         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
481
482 -- Show that the parser is case sensitive
483 testE36 :: Test
484 testE36 = let s = "Viewdef A 1 2"
485         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
486 testE37 :: Test
487 testE37 = let s = "Group A [ B ]"
488         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
489 testE38 :: Test
490 testE38 = let s = "rectangle a 1 2 3 4 Orange"
491         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
492
493 -- Use negative values
494 testE39 :: Test
495 testE39 = let s = "viewdef A -2 5"
496         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
497 testE40 :: Test
498 testE40 = let s = "viewdef A (-2) 5"
499         in TestCase $ assertEquals "" (Left $ NoParsePossible s) (parseString s)
500
501 -- Show that simple expression can create negative values to by pass the
502 -- integer restriction
503 testE41 :: Test
504 testE41 = let s = "viewdef A (0-2) 5"
505         in TestCase $ assertEquals ""
506             (Right [Def $ Viewdef "A" (Minus (Const 0) (Const 2)) (Const 5)])

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

507         (parseString s)
508
509 -- Show the UnexpectedRemainder error
510 -- Umm how?
511
512 -----
513 ----- Unit tests -----
514 ----- Parse file tests -----
515 -----
516
517 -- Parse empty file
518 testF1 :: IO Bool
519 testF1 = testFile "test_files/empty.salsa"
520
521 -- Parse simple salsa
522 testF2 :: IO Bool
523 testF2 = testFile "test_files/simple.salsa"
524
525 -- Parse multi salsa
526 testF3 :: IO Bool
527 testF3 = testFile "test_files/multi.salsa"
528
529 -- Parse invalid salsa
530 testF4 :: IO Bool
531 testF4 = testFile "test_files/invalid.salsa"
532
533 -- Parse non-existing file
534 -- Note that if the file does exist, then it returns the parse from here
535 -- So this does only test the intended if the file indeed does not exist.
536 testF5 :: IO Bool
537 testF5 = catch (testFile "test_files/doesNotExist.salsa")
538              (\e -> do let _ = e :: IOException
539                      return True)
540
541 -- A helper function to compare the results from parseString and parseFile
542 testFile :: String -> IO Bool
543 testFile path_ = do
544   content <- readFile path_
545   output_ <- parseFile path_
546   return $ parseString content == output_

```

Assignment Project Exam Help
<https://powcoder.com>
 Add WeChat powcoder

9.10 multi.salsa

```

1 viewdef One 500 500
2 viewdef Two 400 400
3 group Both [One Two]
4 view Both
5 rectangle larry 10 350 20 20 blue
6 rectangle fawn 300 350 15 25 plum
7
8 view Two
9 larry -> (300, 350) || fawn -> (10,350)
10
11 view Both
12 larry fawn -> +(0, 0 - 300)

```

9.11 simple.salsa

```
1 viewdef Default 400 400
2 rectangle box 10 400 20 20 green
3 box -> (10, 200)
4 box -> +(100, 0)
5 box -> (110,400)
6 box -> +(0-100, 0)
```

9.12 empty.salsa

9.13 invalid.salsa

```
1 viewdef Default 400 a
2 rectangle Box 10 assda 20 20 green 10 23
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder