

EXAM REPORT

Advanced Programming 2013

Rasmus Borgsmidt

CONTENTS

LISTINGS	2
LIST OF TABLES	2
INTRODUCTION	2
1 SALSA LANGUAGE PARSER	2
1.1 Choice of Parser Library	2
1.2 Grammar Transformations	2
1.3 Testing	3
1.4 Assessment	3
2 SALSA LANGUAGE INTERPRETER	3
3 ATOMIC TRANSACTION SERVER	4
4 CONCLUSION	5
BIBLIOGRAPHY	5
A QUESTION 1 CODE FILES	7
A.1 Source Code	7
A.2 Test Code	10
A.3 Test Output	13
A.4 Used Hand-outs	15
A.5 Sample Files	17
B QUESTION 2 CODE FILES	19
B.1 Source Code	19
B.2 Test Code	25
B.3 Used Hand-outs	27
B.4 Sample Files	27

C	QUESTION 3 CODE FILES	28
C.1	Source Code	28
C.2	Test Code	37
C.3	Test Output	40
D	SUBMITTED FILE TREE	42

LISTINGS

1	src/salsa/SalsaParser.hs	7
2	src/salsa/SalsaParserTest.hs	10
3	Session output: src/salsa/SalsaParserTest	13
4	src/salsa/SalsaAst.hs	15
5	src/salsa/SimpleParse.hs	15
6	src/salsa/simple.salsa	17
7	src/salsa/multi.salsa	17
8	src/salsa/SalsaInterp.hs	19
9	src/salsa/SalsaInterpTest.hs	25
10	src/at_server/at_server.erl	28
11	src/at_server/at_trans.erl	32
12	src/at_server/at_extapi.erl	34
13	src/at_server/at_server_tests.erl	37
14	src/at_server/at_extapi_tests.erl	39
15	Session output: src/at_server/at_server_tests.erl	40
16	Session output: src/at_server/at_extapi_tests.erl	41
17	Session output: commit_t_competing_test() (debug enabled)	41
18	File tree under src/	42

LIST OF TABLES

1	LL(1) SALSA Grammar	6
---	---------------------	---

INTRODUCTION

1 SALSA LANGUAGE PARSER

1.1 *Choice of Parser Library*

1.2 *Grammar Transformations*

Goal: to make the grammar LL(1).

Emphasis in correctness. Code can be improved for efficiency later. Fx. using many1 etc.

Assumptions: `||` is left-associative. The original grammar does not specify this, This was chosen because of symmetri with '@'.

'+' and '-' have the same precedence for expressions

As outlined by Mogensen [1, p. 69].

1. Eliminate ambiguity
2. Eliminate left-recursion
3. Perform left-factorization where required

Note: And it is only required when several productions *for the same nonterminal* begin with the same sequence of symbols; for example, it is not a problem that both *Pos* and *Prim* have a production that begins with '(' , as long as they have only one each.

The resulting grammar makes it explicit that '@' has higher precedence than '||'. In other words, it makes it impossible to derive the wrong parse tree from a given input.

1.3 Testing

Do not want to expose functions from module in order to test. But still come up with strategy for somewhat orthogonal testing.

1.4 Assessment

I believe this is a complete and correct solution to question 1.

Allow trailing whitespace SIdents and VIdents parsers could be parameterized and combined.

Comment on that it is particularly tricky to do identifiers separated by space.

extensive test suite machine-like translation from grammar well-chosen grammar transformations

2 SALSA LANGUAGE INTERPRETER

A context comprises an environment and a state; the environment specifies views, shapes and other definitions, the state specifies the position of each shape on the views. In that sense, a given state specifies a *key frame* and the transition from each key frame to the next is specified by the difference between one state and the next.

Apart from possible implications on ordering of graphics instructions, will it impact the resulting Animation if our interpreter executes concurrent commands in parallel or sequentially?

I don't think so, given the

If views are defined later in the animation, they just stay blank until something happens on them.

Comment on the point of the separate SalsaCommand monad, perhaps to make it clear that commands cannot impact the environment.

SalsaCommand is responsible for generating frames – why else would the frame rate be required in the context?

Explain why this is the case (concurrent execution - other reasons?): - NOTE: Two sequenced SalsaCommands are deliberately executed in the same context, - the state is not updated before the second execution -

Unnecessary to nest environment inside context – as long as commands can only amend state.

In a blank context, there should be a blank frame from the outset. Explain how the frames (and frame sets) are manipulated. Why are we using frame sets?

I feel that adding a new shape onto the frame should also be a SalsaCommand (explain why it is not)

Talk about not using more monads – eval is too simple

– When talking about this type, point out that no error handling is necessary – This type reflects that running a command cannot update the environment, – just the state – The type captures the effect of a command in a given context newtype SalsaCommand a = SalsaCommand runSC :: Context -> (a, State)

3 ATOMIC TRANSACTION SERVER

Why have I chosen to use OTP? Talk about time outs and client-provided query functions

Using a waiting list of clients is safe because query_t is blocking so each client can appear at most once in the list

clients can build a timeout into the query and update functions

Copy in text from test file

theoretic problem in choiceUpdate with many processes if choiceUpdate is still creating new processes while the first has already completed.

From the exam text:

"[...] the result of choiceUpdate is the result of this commit."

In a strict sense, the "result of this commit"

The direct return value of

4 CONCLUSION

BIBLIOGRAPHY

- [1] Torben Æ. Mogensen. *Introduction to Compiler Design*. Springer-Verlag London Ltd., 2011.

Table 1: LL(1) SALSA Grammar

<i>Program</i>	::=	<i>DefComs</i>
<i>DefComs</i>	::=	<i>DefCom DefComs*</i>
<i>DefComs*</i>	::=	<i>DefCom DefComs*</i>
		ϵ
<i>DefCom</i>	::=	<i>Command</i>
		<i>Definition</i>
<i>Definition</i>	::=	'viewdef' <i>VIdent Expr Expr</i>
		'rectangle' <i>SIdent Expr Expr Expr Expr Colour</i>
		'circle' <i>SIdent Expr Expr Expr Colour</i>
		'view' <i>VIdent</i>
		'group' <i>VIdent '[' VIdents ']'</i>
<i>Command</i>	::=	<i>Command1 Command*</i>
<i>Command*</i>	::=	' ' <i>Command1 Command*</i>
		ϵ
<i>Command1</i>	::=	<i>Command2 Command1*</i>
<i>Command1*</i>	::=	'@' <i>VIdent Command1*</i>
		ϵ
<i>Command2</i>	::=	<i>SIdents '->' Pos</i>
		'{' <i>Command</i> '}'
<i>VIdents</i>	::=	<i>VIdent VIdents*</i>
<i>VIdents*</i>	::=	<i>VIdent VIdents*</i>
		ϵ
<i>SIdents</i>	::=	<i>SIdent SIdents*</i>
<i>SIdents*</i>	::=	<i>SIdent SIdents*</i>
		ϵ
<i>Pos</i>	::=	'(' <i>Expr</i> ',' <i>Expr</i> ')'
		'+' '(' <i>Expr</i> ',' <i>Expr</i> ')'
<i>Expr</i>	::=	<i>Prim Expr*</i>
<i>Expr*</i>	::=	'+' <i>Prim Expr*</i>
		'-' <i>Prim Expr*</i>
		ϵ
<i>Prim</i>	::=	<i>integer</i>
		<i>SIdent</i> '.' <i>Coord</i>
		'(' <i>Expr</i> ')'
<i>Coord</i>	::=	'x' 'y'
<i>Colour</i>	::=	'blue' 'plum' 'red' 'green' 'orange'

A QUESTION 1 CODE FILES

A.1 Source Code

Listing 1: src/salsa/SalsaParser.hs

```
1  --
2  -- Skeleton for Salsa parser
3  -- To be used at the exam for Advanced Programming, B1-2013
4  --
5
6  module SalsaParser
7  (
8      Program
9      , Error
10     , parseString
11     , parseFile
12     , reserved
13     ) where
14
15  import SalsaAst
16  import SimpleParse
17  import Data.Char (isLetter, isDigit, isUpper, isLower)
18
19  -- A string is used to signal an error
20  type Error = String
21
22  -- Reserved words
23  reserved :: [String]
24  reserved = ["viewdef", "rectangle", "circle", "view", "group",
25             "blue", "plum", "red", "green", "orange"]
26  isReserved :: String -> Bool
27  isReserved w = w `elem` reserved
28
29  -- top-level parsing function that returns a program, or a string in case of failure
30  parseString :: String -> Either Error Program
31  parseString input = let res = parse (do r <- program
32                                     spaces -- allows trailing whitespace in a program
33                                     eof
34                                     return r) input
35                      in case res of
36                        [] -> Left ("unable to parse input: " ++ input)
37                        (r,_) -> Right r
38
39  -- top-level parsing function that reads its input from a file
40  parseFile :: FilePath -> IO (Either Error Program)
41  parseFile path = do input <- readFile path
42                    return $ parseString input
43
44  -- Program parser
45  program :: Parser Program
46  program = defComs
47
48  -- DefComs parser
49  defComs :: Parser [DefCom]
50  defComs = do d <- defCom
51            ds <- defComs_
52            return (d:ds)
53
54  -- DefComs* parser
55  defComs_ :: Parser [DefCom]
56  defComs_ = (do d <- defCom
57              ds <- defComs_
58              return (d:ds))
59  <|> return []
60
61  -- DefCom parser
62  defCom :: Parser DefCom
```

```

63 defCom = (do d <- def
64             return (Def d))
65         <|> (do c <- com
66             return (Com c))
67
68 -- Definition parser
69 def :: Parser Definition
70 def = (do symbol "viewdef"
71         v <- vIdent
72         e0 <- expr
73         e1 <- expr
74         return (Viewdef v e0 e1))
75     <|> (do symbol "rectangle"
76         s <- sIdent
77         e0 <- expr
78         e1 <- expr
79         e2 <- expr
80         e3 <- expr
81         c <- col
82         return (Rectangle s e0 e1 e2 e3 c))
83     <|> (do symbol "circle"
84         s <- sIdent
85         e0 <- expr
86         e1 <- expr
87         e2 <- expr
88         c <- col
89         return (Circle s e0 e1 e2 c))
90     <|> (do symbol "view"
91         v <- vIdent
92         return (View v))
93     <|> (do symbol "group"
94         v <- vIdent
95         schar '['
96         vs <- vIdents
97         schar ']'
98         return (Group v vs))
99
100 -- Command parser
101 com :: Parser Command
102 com = com1 >=> com_
103
104 -- Command* parser
105 com_ :: Command -> Parser Command
106 com_ c0 = (do symbol "||"
107             c1 <- com1
108             com_ (Par c0 c1))
109         <|> return c0
110
111 -- Command1 parser
112 com1 :: Parser Command
113 com1 = com2 >=> com1_
114
115 -- Command1* parser
116 com1_ :: Command -> Parser Command
117 com1_ c = (do schar '@'
118             v <- vIdent
119             com1_ (At c v))
120         <|> return c
121
122 -- Command2 parser
123 com2 :: Parser Command
124 com2 = (do ss <- sIdents
125         symbol "->"
126         p <- pos
127         return (Move ss p))
128     <|> (do schar '{'
129         c <- com
130         schar '}'
131         return c)
132
133 -- VIdents parser
134 vIdents :: Parser [Ident]

```



```

135 vIdsents = do v <- vIdent
136             vs <- vIdsents_
137             return (v:vs)
138
139 -- VIdsents* parser
140 vIdsents_ :: Parser [Ident]
141 vIdsents_ = (do many1 space -- identifiers must be separated by whitespace
142             v <- vIdent
143             vs <- vIdsents_
144             return (v:vs))
145 <|> return []
146
147 -- SIdsents parser
148 sIdsents :: Parser [Ident]
149 sIdsents = do s <- sIdent
150             ss <- sIdsents_
151             return (s:ss)
152
153 -- SIdsents* parser
154 sIdsents_ :: Parser [Ident]
155 sIdsents_ = (do many1 space -- identifiers must be separated by whitespace
156             s <- sIdent
157             ss <- sIdsents_
158             return (s:ss))
159 <|> return []
160
161 -- Pos parser
162 pos :: Parser Pos
163 pos = (do schar '('
164         e0 <- expr
165         schar ','
166         e1 <- expr
167         schar ')')
168     return (Abs e0 e1)
169 <|> (do schar '+'
170         schar '('
171         e0 <- expr
172         schar ','
173         e1 <- expr
174         schar ')')
175     return (Rel e0 e1)
176
177 -- Expr parser
178 expr :: Parser Expr
179 expr = prim >=> expr_
180
181 -- Expr* parser
182 expr_ :: Expr -> Parser Expr
183 expr_ e0 = (do schar '+'
184             e1 <- prim
185             expr_ (Plus e0 e1))
186 <|> (do schar '-'
187         e1 <- prim
188         expr_ (Minus e0 e1))
189 <|> return e0
190
191 -- Prim parser
192 prim :: Parser Expr
193 prim = (do i <- integer
194         return (Const i))
195 <|> (do s <- sIdent
196         schar '.'
197         proj s)
198 <|> (do schar '('
199         e <- expr
200         schar ')')
201     return e)
202
203 -- This parser function handles the coordinate selection in Prim expressions
204 proj :: Ident -> Parser Expr
205 proj s = (do schar 'x'
206            return (Xproj s))

```

```

207         <|> (do schar 'y'
208             return (Yproj s))
209
210 -- Colour parser
211 col :: Parser Colour
212 col = (symbol "blue" >> return Blue)
213       <|> (symbol "plum" >> return Plum)
214       <|> (symbol "red" >> return Red)
215       <|> (symbol "green" >> return Green)
216       <|> (symbol "orange" >> return Orange)
217
218 -- integers
219 integer :: Parser Integer
220 integer = token (do intstr <- many1 $ satisfy isDigit
221                  return (read intstr))
222
223 -- identifiers
224 vIdent :: Parser Ident
225 vIdent = ident isUpper
226
227 sIdent :: Parser Ident
228 sIdent = ident isLower
229
230 ident :: (Char -> Bool) -> Parser Ident
231 ident leading = token (do c <- satisfy leading
232                          cs <- letdigs
233                          if (c:cs) `elem` reserved
234                          then reject
235                          else return (c:cs))
236   where letter = satisfy isLetter
237         digit = satisfy isDigit
238         letdigs = many (letter <|> digit <|> char '_')

```

A.2 Test Code

Listing 2: src/salsa/SalsaParserTest.hs

```

1 import Control.Exception (assert)
2 import SalsaParser
3 import SalsaAst
4
5 -- parses a string and compares the AST to an expected program
6 run :: String -> Program -> String
7 run s p = assert (Right p == parseString s) (shw "pass" s)
8
9 -- parses a string and checks that an error occurs
10 err :: String -> Program -> String
11 err s _ = assert (isError $ parseString s) (shw "error (expected)" s)
12
13 -- indicates if a given result is an error
14 isError :: Either String Program -> Bool
15 isError (Left _) = True
16 isError (Right _) = False
17
18 -- formats test output
19 shw :: String -> String -> String
20 shw pre "" = pre ++ ": " ++ "<empty string>"
21 shw pre s = pre ++ ": " ++ s
22
23 -- main runs the test suite
24 main = do putStrLn "\n*** Checking Colour ***"
25         putStrLn $ checkCol run "blue" Blue
26         putStrLn $ checkCol run "plum" Plum
27         putStrLn $ checkCol run "red" Red
28         putStrLn $ checkCol run "green" Green
29         putStrLn $ checkCol run "orange" Orange
30         putStrLn $ checkCol err "violet" Red
31         putStrLn $ checkCol err "green" Green
32         putStrLn $ checkCol err "Blue" Blue

```

```

33 putStrLn $ checkCol err "blue1" Blue
34
35 putStrLn "\n*** Checking Prim ***"
36 putStrLn $ checkExpr run "0" (Const 0)
37 putStrLn $ checkExpr run "42" (Const 42)
38 putStrLn $ checkExpr run "999999999" (Const 999999999)
39 putStrLn $ checkExpr run "(((42)))" (Const 42)
40 putStrLn $ checkExpr run "john . x" (Xproj "john")
41 putStrLn $ checkExpr run "john . y" (Yproj "john")
42 putStrLn $ checkExpr err "-5" (Const 0)
43 putStrLn $ checkExpr err "-5" (Const 0)
44 putStrLn $ checkExpr err "42.2" (Const 0)
45 putStrLn $ checkExpr err ".8" (Const 0)
46
47 putStrLn "\n*** Checking Expr ***"
48 putStrLn $ checkExpr run "1 + 2" (Plus (Const 1) (Const 2))
49 putStrLn $ checkExpr run "1 - 2" (Minus (Const 1) (Const 2))
50 putStrLn $ checkExpr run "1 + 2 + 3"
51   (Plus (Plus (Const 1) (Const 2)) (Const 3))
52 putStrLn $ checkExpr run "1 - 2 - 3"
53   (Minus (Minus (Const 1) (Const 2)) (Const 3))
54 putStrLn $ checkExpr run "1 + 2 - 3"
55   (Minus (Plus (Const 1) (Const 2)) (Const 3))
56 putStrLn $ checkExpr run "1 - 2 + 3"
57   (Plus (Minus (Const 1) (Const 2)) (Const 3))
58
59 putStrLn "\n*** Checking Pos ***"
60 putStrLn $ checkPos run "{0, 0}" (Abs (Const 0) (Const 0))
61 putStrLn $ checkPos run "+ (0, 0)" (Rel (Const 0) (Const 0))
62
63 putStrLn "\n*** Checking SIds ***"
64 putStrLn $ checkSIds run "a" ["a"]
65 putStrLn $ checkSIds run "aBC" ["aBC"]
66 putStrLn $ checkSIds run "a12T" ["a12T"]
67 putStrLn $ checkSIds run "aa bb cc" ["aa", "bb", "cc"]
68 putStrLn $ checkSIds err "1abc" []
69 putStrLn $ checkSIds err "Abc" []
70 putStrLn $ checkSIds err "_abc" []
71 putStrLn $ checkSIds err "-" []
72 mapM_ putStrLn $ (map $ flip (checkSIds err) []) reserved
73
74 putStrLn "\n*** Checking VIds ***"
75 putStrLn $ checkVIds run "A" ["A"]
76 putStrLn $ checkVIds run "Abc" ["Abc"]
77 putStrLn $ checkVIds run "A12t" ["A12t"]
78 putStrLn $ checkVIds run "AA BB CC" ["AA", "BB", "CC"]
79 putStrLn $ checkVIds err "1Abc" []
80 putStrLn $ checkVIds err "aBC" []
81 putStrLn $ checkVIds err "_ABC" []
82 putStrLn $ checkVIds err "-" []
83
84 putStrLn "\n*** Checking Command ***"
85 putStrLn $ checkCommand run "a->(0, 0)"
86   (Move ["a"] (Abs (Const 0) (Const 0)))
87 putStrLn $ checkCommand run "a->(0, 0)@V"
88   (At (Move ["a"] (Abs (Const 0) (Const 0))) "V")
89 putStrLn $ checkCommand run "a->(0, 0)@V@W"
90   (At (At (Move ["a"] (Abs (Const 0) (Const 0))) "V") "W")
91 putStrLn $ checkCommand run "a->(0, 0)||b->(0, 0)"
92   (Par (Move ["a"] (Abs (Const 0) (Const 0)))
93     (Move ["b"] (Abs (Const 0) (Const 0))))
94 putStrLn $ checkCommand run "a->(0, 0)||b->(0, 0)||c->(0, 0)"
95   (Par (Par (Move ["a"] (Abs (Const 0) (Const 0)))
96     (Move ["b"] (Abs (Const 0) (Const 0))))
97     (Move ["c"] (Abs (Const 0) (Const 0))))
98 putStrLn $ checkCommand run "a->(0, 0)||b->(0, 0)@V"
99   (Par (Move ["a"] (Abs (Const 0) (Const 0)))
100     (At (Move ["b"] (Abs (Const 0) (Const 0))) "V"))
101 putStrLn $ checkCommand run "{a->(0, 0)||b->(0, 0)}@V"
102   (At (Par (Move ["a"] (Abs (Const 0) (Const 0)))
103     (Move ["b"] (Abs (Const 0) (Const 0)))) "V")
104 putStrLn $ checkCommand run "{a->(0, 0)}"

```

```

105         (Move ["a"] (Abs (Const 0) (Const 0)))
106
107     putStrLn "\n*** Checking Definition ***"
108     putStrLn $ checkDefinition run "viewdef V 0 0" (Viewdef "V" (Const 0) (Const 0))
109     putStrLn $ checkDefinition run "rectangle r 0 0 0 0 blue"
110         (Rectangle "r" (Const 0) (Const 0) (Const 0) (Const 0) Blue)
111     putStrLn $ checkDefinition run "circle c 0 0 0 blue"
112         (Circle "c" (Const 0) (Const 0) (Const 0) Blue)
113     putStrLn $ checkDefinition run "view V" (View "V")
114     putStrLn $ checkDefinition run "group G [X Y Z]" (Group "G" ["X", "Y", "Z"])
115     putStrLn $ checkDefinition err "view1 V" (View "V")
116
117     putStrLn "\n*** Checking Program ***"
118     putStrLn $ err "" []
119
120     putStrLn $ run ("viewdef Default 400 400\n" ++
121         "rectangle box 10 400 20 20 green\n" ++
122         "box -> (10, 200)\n" ++
123         "box -> +(100, 0)\n" ++
124         "box -> (110,400)\n" ++
125         "box -> +(0-100, 0)\n")
126
127         [ Def (Viewdef "Default" (Const 400) (Const 400))
128         , Def (Rectangle "box" (Const 10) (Const 400)
129             (Const 20) (Const 20) Green)
130         , Com (Move ["box"] (Abs (Const 10) (Const 200)))
131         , Com (Move ["box"] (Rel (Const 100) (Const 0)))
132         , Com (Move ["box"] (Abs (Const 110) (Const 400)))
133         , Com (Move ["box"] (Rel (Minus (Const 0) (Const 100)) (Const 0)))]
134
135     putStrLn $ run ("viewdef One 500 500\n" ++
136         "viewdef Two 400 400\n" ++
137         "group Both [One Two]\n" ++
138         "view Both\n" ++
139         "rectangle larry 10 350 20 20 blue\n" ++
140         "rectangle fawn 300 350 15 25 plum\n" ++
141         "view Two\n" ++
142         "larry -> (300, 350) || fawn -> (10,350)\n" ++
143         "view Both\n" ++
144         "larry fawn -> +(0, 0 - 300)")
145
146         [ Def (Viewdef "One" (Const 500) (Const 500))
147         , Def (Viewdef "Two" (Const 400) (Const 400))
148         , Def (Group "Both" ["One","Two"])
149         , Def (View "Both")
150         , Def (Rectangle "larry" (Const 10) (Const 350)
151             (Const 20) (Const 20) Blue)
152         , Def (Rectangle "fawn" (Const 300) (Const 350)
153             (Const 15) (Const 25) Plum)
154         , Def (View "Two")
155         , Com (Par (Move ["larry"] (Abs (Const 300) (Const 350)))
156             (Move ["fawn"] (Abs (Const 10) (Const 350))))
157         , Def (View "Both")
158         , Com (Move ["larry","fawn"]
159             (Rel (Const 0) (Minus (Const 0) (Const 300))))]
160
161     putStrLn "\n*** Checking parseFile ***"
162     Right ast <- parseFile "multi.salsa"
163     contents <- readFile "multi.salsa"
164     putStrLn $ flip assert (shw "pass" contents) (ast ==
165         [ Def (Viewdef "One" (Const 500) (Const 500))
166         , Def (Viewdef "Two" (Const 400) (Const 400))
167         , Def (Group "Both" ["One","Two"])
168         , Def (View "Both")
169         , Def (Rectangle "larry" (Const 10) (Const 350)
170             (Const 20) (Const 20) Blue)
171         , Def (Rectangle "fawn" (Const 300) (Const 350)
172             (Const 15) (Const 25) Plum)
173         , Def (View "Two")
174         , Com (Par (Move ["larry"] (Abs (Const 300) (Const 350)))
175             (Move ["fawn"] (Abs (Const 10) (Const 350))))
176         , Def (View "Both")

```

```

177         , Com (Move ["larry","fawn"]
178             (Rel (Const 0) (Minus (Const 0) (Const 300))))))
179
180     putStrLn "\n*** All tests completed successfully ***\n"
181
182
183     where
184         checkCol f s c = f ("circle c 0 0 0 " ++ s)
185             [Def (Circle "c" (Const 0) (Const 0) (Const 0) c)]
186         checkExpr f s e = f ("a -> (" ++ s ++ ", " ++ s ++ ")") [Com (Move ["a"] (Abs e e))]
187         checkPos f s p = f ("a -> " ++ s) [Com (Move ["a"] p)]
188         checkSIdent f s ids = f (s ++ " -> (0, 0)")
189             [Com (Move ids (Abs (Const 0) (Const 0)))]
190         checkVIdent f s ids = f ("group V [" ++ s ++ "]") [Def (Group "V" ids)]
191         checkCommand f s c = f s [Com c]
192         checkDefinition f s d = f s [Def d]

```

A.3 Test Output

Listing 3: Session output: src/salsa/SalsaParserTest

```

1 *** Checking Colour ***
2 pass: circle c 0 0 0 blue
3 pass: circle c 0 0 0 plum
4 pass: circle c 0 0 0 red
5 pass: circle c 0 0 0 green
6 pass: circle c 0 0 0 orange
7 error (expected): circle c 0 0 0 violet
8 error (expected): circle c 0 0 0 green
9 error (expected): circle c 0 0 0 Blue
10 error (expected): circle c 0 0 0 blue1
11
12 *** Checking Prim ***
13 pass: a -> (0, 0)
14 pass: a -> (42, 42)
15 pass: a -> (999999999, 999999999)
16 pass: a -> (((42))), (((42)))
17 pass: a -> (john . x, john . x)
18 pass: a -> (john . y, john . y)
19 error (expected): a -> (-5, -5)
20 error (expected): a -> (-5, -5)
21 error (expected): a -> (42.2, 42.2)
22 error (expected): a -> (.8, .8)
23
24 *** Checking Expr ***
25 pass: a -> (1 + 2, 1 + 2)
26 pass: a -> (1 - 2, 1 - 2)
27 pass: a -> (1 + 2 + 3, 1 + 2 + 3)
28 pass: a -> (1 - 2 - 3, 1 - 2 - 3)
29 pass: a -> (1 + 2 - 3, 1 + 2 - 3)
30 pass: a -> (1 - 2 + 3, 1 - 2 + 3)
31
32 *** Checking Pos ***
33 pass: a -> (0, 0)
34 pass: a -> + (0, 0)
35
36 *** Checking SIdents ***
37 pass: a -> (0, 0)
38 pass: aBC -> (0, 0)
39 pass: a12T -> (0, 0)
40 pass: aa bb cc -> (0, 0)
41 error (expected): 1abc -> (0, 0)
42 error (expected): Abc -> (0, 0)
43 error (expected): _abc -> (0, 0)

```

```

44 error (expected): _ -> (0, 0)
45 error (expected): viewdef -> (0, 0)
46 error (expected): rectangle -> (0, 0)
47 error (expected): circle -> (0, 0)
48 error (expected): view -> (0, 0)
49 error (expected): group -> (0, 0)
50 error (expected): blue -> (0, 0)
51 error (expected): plum -> (0, 0)
52 error (expected): red -> (0, 0)
53 error (expected): green -> (0, 0)
54 error (expected): orange -> (0, 0)
55
56 *** Checking Vidents ***
57 pass: group V [A]
58 pass: group V [Abc]
59 pass: group V [A12t]
60 pass: group V [AA BB CC]
61 error (expected): group V [1Abc]
62 error (expected): group V [aBC]
63 error (expected): group V [_ABC]
64 error (expected): group V [_]
65
66 *** Checking Command ***
67 pass: a->(0, 0)
68 pass: a->(0, 0)@V
69 pass: a->(0, 0)@V@W
70 pass: a->(0, 0)||b->(0, 0)
71 pass: a->(0, 0)||b->(0, 0)||c->(0,0)
72 pass: a->(0, 0)||b->(0, 0)@V
73 pass: {a->(0, 0)||b->(0, 0)}@V
74 pass: {{a->(0, 0)}}
75
76 *** Checking Definition ***
77 pass: viewdef V 0 0
78 pass: rectangle r 0 0 0 0 blue
79 pass: circle c 0 0 0 blue
80 pass: view V
81 pass: group G [X Y Z]
82 error (expected): view1 V
83
84 *** Checking Program ***
85 error (expected): <empty string>
86 pass: viewdef Default 400 400
87 rectangle box 10 400 20 20 green
88 box -> (10, 200)
89 box -> +(100, 0)
90 box -> (110,400)
91 box -> +(0-100, 0)
92
93 pass: viewdef One 500 500
94 viewdef Two 400 400
95 group Both [One Two]
96 view Both
97 rectangle larry 10 350 20 20 blue
98 rectangle fawn 300 350 15 25 plum
99
100 view Two
101 larry -> (300, 350) || fawn -> (10,350)
102
103 view Both
104 larry fawn -> +(0, 0 - 300)
105
106 *** Checking parseFile ***
107 pass: viewdef One 500 500
108 viewdef Two 400 400

```

```

109 group Both [One Two]
110 view Both
111 rectangle larry 10 350 20 20 blue
112 rectangle fawn 300 350 15 25 plum
113
114 view Two
115 larry -> (300, 350) || fawn -> (10,350)
116
117 view Both
118 larry fawn -> +(0, 0 - 300)
119
120 *** All tests completed successfully ***

```

A.4 Used Hand-outs

Listing 4: src/salsa/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 = Viewdef Ident Expr Expr
8                 | Rectangle 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 = Move [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

```

Listing 5: src/salsa/SimpleParse.hs

```

1 {-
2   Example code from Advanced Programming lecture.
3
4   Small monadic parser combinator library.
5
6   Date: Sep 20, 2012
7   Author: Ken Friis Larsen <kflarsen@diku.dk>
8 -}
9 module SimpleParse where
10
11 import Control.Monad(MonadPlus(..))
12 import Data.Char (isSpace)
13
14 newtype Parser a = Parser (String -> [(a, String)])
15 parse (Parser p) = p
16
17 parse' p s = [ result | (result,rest) <- parse p s, null rest ]
18

```

```

19
20
21 item :: Parser Char    -- String -> [(Char,String)]
22 item = Parser item'
23   where item' "" = [ ]
24         item' (x : xs) = [(x,xs)]
25
26 reject :: Parser a
27 reject = Parser $ \ _ -> []
28
29 eof :: Parser ()
30 eof = Parser eof'
31   where eof' "" = [(),[]]
32         eof' _ = []
33
34 parseEof p = parse $ p >> eof >= return . fst
35
36
37 (>>) :: Parser a -> Parser b -> Parser (a,b)
38 p >> q = Parser $ \ s -> [ ((a,b), cs) | (a, cs1) <- parse p s
39                                     , (b, cs) <- parse q cs1]
40
41 instance Monad Parser where
42   p >>= q = Parser$ \cs -> [(v2, cs2) |
43                               (v1, cs1) <- parse p cs,
44                               (v2, cs2) <- parse (q v1) cs1]
45
46   return v = Parser$ \cs -> [(v, cs)]
47
48
49 (<++) :: Parser a -> Parser a -> Parser a
50 p <++ q = Parser (\cs -> case parse p cs of
51                           [] -> parse q cs
52                           res -> res)
53
54
55
56 char :: Char -> Parser Char
57 char e = do c <- item
58           if e == c
59             then return c
60             else reject
61
62 satisfy :: (Char -> Bool) -> Parser Char
63 satisfy p = do c <- item
64             if p c
65               then return c
66               else reject
67
68 string :: String -> Parser String
69 string "" = return ""
70 string (c:cs) = do char c
71                  string cs
72                  return (c:cs)
73
74 (<|>) :: Parser a -> Parser a -> Parser a
75 p <|> q = Parser$ \cs -> parse p cs ++ parse q cs
76
77 instance MonadPlus Parser where
78   p `mplus` q = p <|> q
79   mzero      = reject
80
81
82 many :: Parser a -> Parser [a]
83 many p = do v <- p
84           vs <- many p
85           return (v:vs)
86           <|> return []
87
88 many1 :: Parser a -> Parser [a]
89 many1 p = do v <- p
90           vs <- many p

```



```

91         return (v:vs)
92
93 sepBy      :: Parser a -> Parser b -> Parser [a]
94 p `sepBy` sep = (p `sepBy1` sep) <|> return []
95
96 sepBy1     :: Parser a -> Parser b -> Parser [a]
97 p `sepBy1` sep = do {a <- p; as <- many (do {sep; p}); return (a:as)}
98
99 chainl     :: Parser a -> Parser (a -> a -> a) -> a -> Parser a
100 chainl p op a = (p `chainl1` op) <|> return a
101
102 chainl1    :: Parser a -> Parser (a -> a -> a) -> Parser a
103 p `chainl1` op = do a <- p
104                   rest a
105                   where
106                     rest a = do f <- op
107                               b <- p
108                               rest (f a b)
109                               <|> return a
110
111
112
113
114 option :: Parser a -> Parser (Maybe a)
115 option p = do v <- p
116             return (Just v)
117             <|> return Nothing
118
119
120 -- Lexical combinators: -----
121
122 space      :: Parser Char
123 space      = satisfy isSpace
124
125 spaces     :: Parser String
126 spaces     = many space
127
128 token      :: Parser a -> Parser a
129 token p    = spaces >> p
130
131 symbol     = token . string
132 schar      = token . char

```

A.5 Sample Files

Listing 6: src/salsa/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)

```

Listing 7: src/salsa/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)`

B QUESTION 2 CODE FILES

B.1 Source Code

Listing 8: src/salsa/SalsaInterp.hs

```
1  --
2  -- Skeleton for Salsa interpreter
3  -- To be used at the exam for Advanced Programming, B1-2013
4  --
5
6  module SalsaInterp (Position, interpolate, runProg)
7  where
8
9  import SalsaAst
10 import Gpx
11 import Data.List(intersect, union, (\\))
12 import qualified Data.Map as M
13
14 type Position = (Integer, Integer)
15
16
17 --
18 -- Primary top-level function
19 --
20
21 runProg :: Integer -> Program -> Animation
22 runProg framerate program =
23   let salsas = map defCom program
24       ((_, context) = runSalsa (sequence_ salsas) $ baseContext framerate
25   in animation context
26
27 --
28 -- The function interpolate
29 --
30
31 interpolate :: Integer -> Position -> Position -> [Position]
32 interpolate rate (x0, y0) (x1, y1)
33   | rate <= 0 = error "framerate must be positive"
34   | otherwise = zip (ipol rate x0 x1) (ipol rate y0 y1)
35
36 ipol :: Integer -> Integer -> Integer -> [Integer]
37 ipol rate start end =
38   let step = (fromIntegral $ end - start) / fromIntegral rate
39   in map ((start +) . round . (* step)) ([1.0..fromIntegral rate] :: [Double])
40
41 --
42 -- Data type representing the two types of shape
43 --
44 -- A shape has an identifier, some shape-specific information, a color name
45 -- and a list of the views it is defined on
46 data Shape = Rect Ident Integer Integer String [ViewT]
47            | Circ Ident Integer Integer String [ViewT]
48            deriving (Show, Eq, Ord)
49
50 idnt :: Shape -> Ident
51 idnt (Rect ident _ _ _) = ident
52 idnt (Circ ident _ _ _) = ident
53
54 vs :: Shape -> [ViewT]
55 vs (Rect _ _ _ vws) = vws
56 vs (Circ _ _ _ vws) = vws
57
58 --
59 -- Data types relating to the (by commands) read-only environment and writable state
60 --
61
62 -- 'T' suffix is just to distinguish from Salsa AST constructors
63 type ViewT = (Ident, Integer, Integer)
64 type GroupT = (Ident, [ViewT])
```

```

65 type ViewMap = M.Map Ident ViewT
66 type ShapeMap = M.Map Ident Shape
67 type GroupMap = M.Map Ident GroupT
68 -- The Context represents all the information, both read-only and writable
69 data Context = Context {
70     views :: ViewMap
71     , shapes :: ShapeMap
72     , groups :: GroupMap
73     , activeViews :: [ViewT]
74     , frameRate :: Integer
75     , state :: State
76     } deriving (Show, Eq)
77
78 type PosMap = M.Map Ident (M.Map ViewT Position)
79 -- An ExtFrame associates each graphics command with the shape it draws
80 -- This information is used to combine frame sets produced by commands that need
81 -- to run in parallel
82 type ExtFrame = [(Ident, GpxInstr)]
83 type FrameSet = [ExtFrame]
84 -- The State data type is used for the writable information
85 -- shapePos is a map of maps, storing the current position of each shape on each view
86 -- A FrameSet stores 'framerate' number of frames, and each frame set holds the
87 -- frames between one key frame (excl) and the next (incl)
88 -- The frames of a frame set are stored in reverse order, so the next key frame
89 -- of a frame set is always at the head of the list
90 -- The frame sets are also held in reverse order with the most recent at the head
91 data State = State {
92     shapePos :: PosMap
93     , frameSets :: [FrameSet]
94     } deriving (Show, Eq)
95
96 blankFrame :: ExtFrame
97 blankFrame = []
98
99 -- The base state has a single frame in a single frame set representing the single
100 -- initial key frame
101 baseState :: State
102 baseState = State M.empty [[blankFrame]]
103
104 -- The base context holds empty maps, the framerate and a base state
105 baseContext :: Integer -> Context
106 baseContext rate = Context M.empty M.empty M.empty [] rate baseState
107
108 --
109 -- functions to query and manipulate the context
110 --
111
112 updateViews :: (ViewMap -> ViewMap) -> Context -> Context
113 updateViews f = \context -> context { views = f (views context) }
114
115 updateShapes :: (ShapeMap -> ShapeMap) -> Context -> Context
116 updateShapes f = \context -> context { shapes = f (shapes context) }
117
118 updateGroups :: (GroupMap -> GroupMap) -> Context -> Context
119 updateGroups f = \context -> context { groups = f (groups context) }
120
121 updateState :: (State -> State) -> Context -> Context
122 updateState f = \context -> context { state = f (state context) }
123
124 updateFrameSets :: ([FrameSet] -> [FrameSet]) -> State -> State
125 updateFrameSets f = \st -> st { frameSets = f (frameSets st) }
126
127 updateShapePos :: (PosMap -> PosMap) -> State -> State
128 updateShapePos f = \st -> st { shapePos = f (shapePos st) }
129
130 -- the key frame of a given context is always the latest frame in the latest frame set
131 -- and a context always has at least a key frame
132 updateKeyFrame :: (ExtFrame -> ExtFrame) -> [FrameSet] -> [FrameSet]
133 updateKeyFrame f = \((kf:set):sets) -> ((f kf):set):sets
134
135 -- Returns the shapes present on the active views
136 activeShapes :: Context -> [Shape]

```

```

137 activeShapes context = let shps = M.elems $ shapes context
138                        vws = activeViews context
139                        in filter (any (`elem` vws) . vs) shps
140
141 -- Helper function that determines which shapes need to move
142 shapesToMove :: [Ident] -> Context -> [Shape]
143 shapesToMove idents context =
144   let moving = lookupObjs "moveShapes" context idents (shapes context)
145       active = activeShapes context
146   in if null $ moving \\ active
147      then moving
148      else error $ "shapes not defined on all active views: " ++ show idents
149          ++ "\ncontext: " ++ show context
150
151 -- Helper function that returns the position of a given shape on a given view
152 shapeViewPos :: Shape -> ViewT -> Context -> Position
153 shapeViewPos shape view context =
154   let posMap = shapePos $ state context
155       pmap = lookupObj "" context (idnt shape) posMap
156   in lookupObj "" context view pmap
157
158 activePos :: Shape -> Context -> [Position]
159 activePos shape context =
160   let vws = activeViews context
161       pmap = lookupObj "activePos" context (idnt shape) $ shapePos $ state context
162   in lookupObjs "activePos" context vws pmap
163
164 updateActivePos :: Maybe Pos -> Shape -> Context -> Context
165 updateActivePos pos shape context =
166   updateState (updateShapePos $ updatePosMap pos shape context) context
167
168 updatePosMap :: Maybe Pos -> Shape -> Context -> PosMap -> PosMap
169 updatePosMap pos shape context =
170   \pmap -> let fromPos = activePos shape context
171             toPos = getToPositions context fromPos pos
172             vws = activeViews context
173             mp = lookupObj "updatePosMap" context (idnt shape) pmap
174             mp' = foldl (\m (k, v) -> M.insert k v m) mp $ zip vws toPos
175             in M.insert (idnt shape) mp' pmap
176
177 addToPosMap :: Ident -> Position -> Context -> PosMap -> PosMap
178 addToPosMap ident pos context =
179   \pmap -> let vws = activeViews context
180             mp = foldl (\m k -> M.insert k pos m) M.empty vws
181             in M.insert ident mp pmap
182
183 -- Returns the Animation produced by this interpreter
184 -- This just flattens the list of frames and reverses it
185 animation :: Context -> Animation
186 animation context =
187   let vws = M.elems $ views context
188       fs = reverse . concat . frameSets $ state context
189   in (vws , map stripFrame fs)
190
191 -- Removes the additional shape information from each frame to produce frames
192 -- suitable for the graphics backend
193 stripFrame :: ExtFrame -> Frame
194 stripFrame pairs = snd $ unzip pairs
195
196
197 --
198 -- Functions generating graphics instructions for moving shapes
199 --
200
201 -- Top-level function that causes the generation of the necessary graphics
202 -- instructions to move a set of shapes to a new absolute or relative position.
203 -- This function is responsible for generating each new frame set.
204 -- Strategy: Generate a complete set of instructions to draw everything in place;
205 -- then overwrite the instructions for the shapes that needs to move
206 moveShapes :: [Ident] -> Pos -> Context -> State
207 moveShapes idents pos context =
208   let moving = shapesToMove idents context

```

```

209     -- 'neutral' map with full set of combinations to draw everything in its
210     -- current position
211     wmap = M.fromList [(s, v), (shapeViewPos s v context, Nothing)] |
212         s <- M.elems (shapes context), v <- vs s]
213     -- list of shapes to move, used to overwrite entries in the map
214     moves = [(s, v), (shapeViewPos s v context, Just pos)] |
215         s <- moving, v <- activeViews context]
216     wmap' = foldl (\m (k, v) -> M.insert k v m) wmap moves
217     fs = replicate (fromIntegral $ frameRate context) blankFrame
218     fs' = foldl (writeToFrameSet context) fs $ M.toList wmap'
219     context' = foldr (updateActivePos $ Just pos) context moving
220     in updateFrameSets (fs' :) $ state context'
221
222 -- Helper function that updates a frame set with instructions for a
223 -- particular shape and view
224 writeToFrameSet :: Context -> FrameSet -> ((Shape, ViewT), (Position, Maybe Pos))
225               -> FrameSet
226 writeToFrameSet context fs ((shape, view), (fromPos, p)) =
227     let toPos = getToPos context fromPos p
228         -- the frames in a frame set is stored in reverse order so the key frame
229         -- is at the front of the list
230         pos = reverse $ interpolate (toInteger $ length fs) fromPos toPos
231     in zipWith (writeToFrame shape view) pos fs
232
233 -- Helper function that writes a single shape on a single view to a single frame
234 writeToFrame :: Shape -> ViewT -> Position -> ExtFrame -> ExtFrame
235 writeToFrame (Rect ident width height colname _) (vname,_,_) (llx, lly) frame =
236     (ident, (DrawRect llx lly width height vname colname)):frame
237 writeToFrame (Circ ident r colname _) (vname,_,_) (x, y) frame =
238     (ident, (DrawCirc x y r vname colname)):frame
239
240 -- Helper functions that determine new positions based on current position
241 -- and absolute/relative position information
242 getToPositions :: Context -> [Position] -> Maybe Pos -> [Position]
243 getToPositions context positions pos = map (\p -> getToPos context p pos) positions
244
245 getToPos :: Context -> Position -> Maybe Pos -> Position
246 getToPos _ fromPos Nothing = fromPos
247 getToPos context _ (Just (Abs xExpr yExpr)) =
248     (eval context xExpr, eval context yExpr)
249 getToPos context (x, y) (Just (Rel xExpr yExpr)) =
250     (x + eval context xExpr, y + eval context yExpr)
251
252 --
253 -- Functions for implementing new definitions in context and current frame
254 --
255
256 addView :: Ident -> ViewT -> Context -> Context
257 addView ident view context =
258     activate ident $ updateViews (M.insert ident view) context
259
260 addShape :: Ident -> Shape -> Position -> Context -> Context
261 addShape ident shape pos context =
262     let context' = updateShapes (M.insert ident shape) context
263         context'' = updateState (updateShapePos $ addToPosMap ident pos context') context'
264     in writeShapeToKeyFrame shape pos context''
265
266 writeShapeToKeyFrame :: Shape -> Position -> Context -> Context
267 writeShapeToKeyFrame shape pos context =
268     let writer = \view frame -> writeToFrame shape view pos frame
269         updater = \frame -> foldr writer frame $ activeViews context
270     in updateState (updateFrameSets $ updateKeyFrame updater) context
271
272 addGroup :: Ident -> [Ident] -> Context -> Context
273 addGroup ident idents context =
274     let vws = lookupObjs "addGroup" context idents (views context)
275     in updateGroups (M.insert ident (ident, vws)) context
276
277 activate :: Ident -> Context -> Context
278 activate ident context =
279 {- I wonder if there is a clever way of stringing together multiple Maybes,
280    - branching on Nothing. It is opposite of the usual Maybe Monad behavior where

```

```

281 -- the occurrence of a single Nothing forces the combined result to Nothing -}
282 case M.lookup ident (views context) of
283   Just view -> context { activeViews = [view] }
284   Nothing -> case M.lookup ident (groups context) of
285     Just (_, vws) -> context { activeViews = vws }
286     Nothing -> error $ "undefined view or group: " ++ show ident
287
288
289 --
290 -- Monad types SalsaCommand and Salsa
291 --
292
293 -- This type reflects that running a command cannot update the environment,
294 -- just the state
295 -- The type captures the effect of a command in a given context, that is a move
296 -- from the current key frame to a new
297 newtype SalsaCommand a = SalsaCommand { runSC :: Context -> (a, State) }
298 instance Monad SalsaCommand where
299   return x = SalsaCommand $ \context -> (x, state context)
300   m >=> f = SalsaCommand $ \context -> let (x, st) = runSC m context
301     in runSC (f x) context { state = st }
302
303 -- The Salsa type represents an animation step; it is either a definition
304 -- activating something or adding a new shape to views on the current key frame,
305 -- or it is a new command that causes the generation of a new frame set from one
306 -- key frame to the next
307 newtype Salsa a = Salsa { runSalsa :: Context -> (a, Context) }
308 instance Monad Salsa where
309   return x = Salsa $ \context -> (x, context)
310   m >=> f = Salsa $ \context -> let (x, context') = runSalsa m context
311     in runSalsa (f x) context'
312
313 -- Changes the context locally for the command
314 local :: (Context -> Context) -> SalsaCommand a -> SalsaCommand a
315 local f m = SalsaCommand $ \context -> runSC m (f context)
316
317 -- Captures the effect of a command in a SalsaCommand
318 command :: Command -> SalsaCommand ()
319 command (At com ident) = local (activate ident) $ command com
320 command (Par com0 com1) = command com0 >> command com1 >> mergeConcurrent com0 com1 ()
321 command (Move idents pos) =
322   SalsaCommand $ \context -> ((), moveShapes idents pos context)
323
324 -- Helper function that captures the effect of running to commands in parallel
325 mergeConcurrent :: Command -> Command -> a -> SalsaCommand a
326 mergeConcurrent com0 com1 x =
327   SalsaCommand $ \context ->
328     (x, updateFrameSets (mergeFrameSets com0 com1) $ state context)
329
330 -- Helper function that merges the two latest frame sets. This is necessary when
331 -- commands should run in parallel and therefore manipulate the same frame set
332 -- Strategy: take the latest frame set (head of list), which was generated by
333 -- com1, and copy across any instruction pertaining to shapes manipulated by com0.
334 mergeFrameSets :: Command -> Command -> [FrameSet] -> [FrameSet]
335 mergeFrameSets com0 com1 (fs1:fs0:sets) =
336   let shps0 = shapesFromCommand com0
337       shps1 = shapesFromCommand com1
338   in if null $ intersect shps0 shps1
339     then (zipWith (mergeFrames shps1 shps0) fs1 fs0):sets
340     else error $ "concurrent commands manipulating same shapes: "
341       ++ show shps0 ++ " and " ++ show shps1
342 mergeFrameSets _ _ _ = error "invalid frame set configuration"
343
344 -- Strategy: In order not to overwrite the wrong instructions, we take the two
345 -- frame sets and remove from each, any instruction pertaining to a shape
346 -- manipulated by the other command. After this, it is safe just to combine the
347 -- frame sets with union
348 mergeFrames :: [Ident] -> [Ident] -> ExtFrame -> ExtFrame -> ExtFrame
349 mergeFrames shps1 shps0 frame1 frame0 =
350   let frame0' = filter (\(ident, _) -> not $ ident `elem` shps1) frame0
351       frame1' = filter (\(ident, _) -> not $ ident `elem` shps0) frame1
352   in union frame1' frame0'

```

```

353
354 -- Helper function that determines the shapes manipulated by a given command.
355 -- This is used when merging frame sets for parallel commands
356 shapesFromCommand :: Command -> [Ident]
357 shapesFromCommand (At com _) = shapesFromCommand com
358 shapesFromCommand (Par com0 com1) = shapesFromCommand com0 ++ shapesFromCommand com1
359 shapesFromCommand (Move ident _) = ident
360
361 -- Recursive evaluation function for the Salsa Expression type
362 eval :: Context -> Expr -> Integer
363 eval _ (Const val) = val
364 eval context (Plus expr0 expr1) = eval context expr0 + eval context expr1
365 eval context (Minus expr0 expr1) = eval context expr0 - eval context expr1
366 eval context (Xproj ident) =
367     let shape = lookupObj "eval Xproj" context ident (shapes context)
368         (xpos, _) = unzip $ activePos shape context
369     in foldl min 0 xpos
370 eval context (Yproj ident) =
371     let shape = lookupObj "eval Yproj" context ident (shapes context)
372         (_, ypos) = unzip $ activePos shape context
373     in foldl min 0 ypos
374
375 -- Captures the effect of a definition in a Salsa computation
376 definition :: Definition -> Salsa ()
377 definition (Viewdef ident wExpr hExpr) =
378     Salsa $ \context -> let width = eval context wExpr
379                          height = eval context hExpr
380                          in ((), addView ident (ident, width, height) context)
381 definition (Rectangle ident llxExpr llyExpr wExpr hExpr col) =
382     Salsa $ \context -> let llx = eval context llxExpr
383                          lly = eval context llyExpr
384                          width = eval context wExpr
385                          height = eval context hExpr
386                          vws = activeViews context
387                          rect = Rect ident width height (colorName col) vws
388                          in ((), addShape ident rect (llx, lly) context)
389 definition (Circle ident xExpr yExpr rExpr col) =
390     Salsa $ \context -> let x = eval context xExpr
391                          y = eval context yExpr
392                          r = eval context rExpr
393                          vws = activeViews context
394                          circle = Circ ident r (colorName col) vws
395                          in ((), addShape ident circle (x, y) context)
396 definition (View ident) =
397     Salsa $ \context -> ((), activate ident context)
398 definition (Group ident ident) =
399     Salsa $ \context -> ((), addGroup ident ident context)
400
401 -- Helper function to wrap a SalsaCommand as a Salsa computation
402 liftC :: SalsaCommand a -> Salsa a
403 liftC sc = Salsa $ \context -> let (x, st) = runSC sc context
404                                in (x, context { state = st })
405
406 -- Helper function to generate a Salsa computation from a DefCom
407 defCom :: DefCom -> Salsa ()
408 defCom (Def def) = definition def
409 defCom (Com com) = liftC $ command com
410
411 --- Other helper functions
412
413 -- looks up objects in a map and throws an error if the element(s) are not there.
414 lookupObjs :: Ord b => String -> Context -> [b] -> M.Map b a -> [a]
415 lookupObjs msg context ident m = map (flip (lookupObj msg context) m) ident
416
417 lookupObj :: Ord b => String -> Context -> b -> M.Map b a -> a
418 lookupObj msg context ident m =
419     case M.lookup ident m of
420         Nothing -> error $ "undefined object: " ++ "\ncontext: " ++ show context ++ msg
421         Just v -> v
422
423 colorName :: Colour -> String
424 colorName Blue = "blue"

```



```

425 colorName Plum = "plum"
426 colorName Red = "red"
427 colorName Green = "green"
428 colorName Orange = "orange"

```

B.2 Test Code

Listing 9: src/salsa/SalsaInterpTest.hs

```

1 [ Def (Viewdef "Default" (Const 400) (Const 400)), Def (Rectangle "box" (Const 10) (Const
  400) (Const 20) (Const 20) Green), Com (Move ["box"] (Abs (Const 10) (Const 200))),
  Com (Move ["box"] (Rel (Const 100) (Const 0))), Com (Move ["box"] (Abs (Const 110) (
  Const 400))), Com (Move ["box"] (Rel (Minus (Const 0) (Const 100)) (Const 0))) ]
2 ([("Default",400,400)],[[DrawRect 10 400 20 20 "Default" "green"],[DrawRect 10 400 20 20 "
  Default" "green"],[DrawRect 10 200 20 20 "Default" "green"],[DrawRect 110 200 20 20 "
  Default" "green"],[DrawRect 110 400 20 20 "Default" "green"]])
3
4 [ Def (Viewdef "One" (Const 500) (Const 500)) , Def (Viewdef "Two" (Const 400) (Const 400)
  ) , Def (Group "Both" ["One","Two"]) , Def (View "Both"), Def (Rectangle "larry" (
  Const 10) (Const 350) (Const 20) (Const 20) Blue), Def (Rectangle "fawn" (Const 300)
  (Const 350) (Const 15) (Const 25) Plum), Def (View "Two") , Com (Par (Move ["larry"]
  (Abs (Const 300) (Const 350))) (Move ["fawn"] (Abs (Const 10) (Const 350)))) , Def (
  View "Both"), Com (Move ["larry","fawn"] (Rel (Const 0) (Minus (Const 0) (Const 300))
  )))
5 ([("One",500,500),("Two",400,400)],
6 [[DrawRect 300 350 15 25 "One" "plum",DrawRect 300 350 15 25 "Two" "plum",DrawRect 10 350
  20 20 "One" "blue",DrawRect 10 350 20 20 "Two" "blue"],
7 [DrawRect 300 350 15 25 "Two" "plum",DrawRect 300 350 15 25 "One" "plum",DrawRect 10 350
  20 20 "Two" "blue",DrawRect 10 350 20 20 "One" "blue"],
8 [DrawRect 300 350 20 20 "Two" "blue",DrawRect 10 350 20 20 "One" "blue",DrawRect 10 350
  15 25 "Two" "plum",DrawRect 300 350 15 25 "One" "plum"]])
9
10 [ Def (Viewdef "One" (Const 500) (Const 500)) , Def (Viewdef "Two" (Const 400) (Const 400)
  ) , Def (Group "Both" ["One","Two"]) , Def (View "Both"), Def (Rectangle "larry" (
  Const 10) (Const 350) (Const 20) (Const 20) Blue), Def (Rectangle "fawn" (Const 300)
  (Const 350) (Const 15) (Const 25) Plum), Com (At (Par (Move ["larry"] (Abs (Const
  300) (Const 350))) (Move ["fawn"] (Abs (Const 10) (Const 350)))) "Two") , Com (Move
  ["larry","fawn"] (Rel (Const 0) (Minus (Const 0) (Const 300)))))
11 ([("One",500,500),("Two",400,400)],
12 [[DrawRect 300 350 15 25 "One" "plum",
13   DrawRect 300 350 15 25 "Two" "plum",
14   DrawRect 10 350 20 20 "One" "blue",
15   DrawRect 10 350 20 20 "Two" "blue"],
16
17   [DrawRect 300 350 15 25 "Two" "plum",
18   DrawRect 300 350 15 25 "One" "plum",
19   DrawRect 10 350 20 20 "Two" "blue",
20   DrawRect 10 350 20 20 "One" "blue"],
21
22   [DrawRect 300 350 20 20 "Two" "blue",
23   DrawRect 10 350 20 20 "One" "blue",
24   DrawRect 10 350 15 25 "Two" "plum",
25   DrawRect 300 350 15 25 "One" "plum"]])
26
27
28 ([("One",500,500),("Two",400,400)],
29 [[DrawRect 300 350 15 25 "One" "plum",
30   DrawRect 300 350 15 25 "Two" "plum",
31   DrawRect 10 350 20 20 "One" "blue",
32   DrawRect 10 350 20 20 "Two" "blue"],
33
34   [DrawRect 300 350 15 25 "Two" "plum",
35   DrawRect 300 350 15 25 "One" "plum",
36   DrawRect 10 350 20 20 "Two" "blue",
37   DrawRect 10 350 20 20 "One" "blue"],
38
39   [DrawRect 300 350 20 20 "Two" "blue",
40   DrawRect 10 350 20 20 "One" "blue",
41   DrawRect 10 350 15 25 "Two" "plum",

```

```

42   DrawRect 300 350 15 25 "One" "plum"]])
43
44
45   ([("One",500,500),("Two",400,400)],
46   [[DrawRect 300 350 15 25 "One" "plum",
47     DrawRect 300 350 15 25 "Two" "plum",
48     DrawRect 10 350 20 20 "One" "blue",
49     DrawRect 10 350 20 20 "Two" "blue"],
50
51     [DrawRect 10 350 15 25 "Two" "plum",
52       DrawRect 300 350 15 25 "One" "plum",
53       DrawRect 300 350 20 20 "Two" "blue",
54       DrawRect 10 350 20 20 "One" "blue"],
55
56     [DrawRect 300 50 20 20 "Two" "blue",
57       DrawRect 10 50 20 20 "One" "blue",
58       DrawRect 10 50 15 25 "Two" "plum",
59       DrawRect 300 50 15 25 "One" "plum"]])
60
61
62
63
64
65
66
67   ([("One",500,500),("Two",400,400)],
68   [[DrawRect 300 350 15 25 "One" "plum",
69     DrawRect 300 350 15 25 "Two" "plum",
70     DrawRect 10 350 20 20 "One" "blue",
71     DrawRect 10 350 20 20 "Two" "blue"],
72
73     [DrawRect 107 350 15 25 "Two" "plum",
74       DrawRect 300 350 15 25 "One" "plum",
75       DrawRect 203 350 20 20 "Two" "blue",
76       DrawRect 10 350 20 20 "One" "blue"],
77
78     [DrawRect 203 350 15 25 "Two" "plum",
79       DrawRect 300 350 15 25 "One" "plum",
80       DrawRect 107 350 20 20 "Two" "blue",
81       DrawRect 10 350 20 20 "One" "blue"],
82
83     [DrawRect 300 350 15 25 "Two" "plum",
84       DrawRect 300 350 15 25 "One" "plum",
85       DrawRect 10 350 20 20 "Two" "blue",
86       DrawRect 10 350 20 20 "One" "blue"],
87
88     [DrawRect 300 150 20 20 "Two" "blue",
89       DrawRect 10 150 20 20 "One" "blue",
90       DrawRect 10 150 15 25 "Two" "plum",
91       DrawRect 300 150 15 25 "One" "plum"],
92
93     [DrawRect 300 250 20 20 "Two" "blue",
94       DrawRect 10 250 20 20 "One" "blue",
95       DrawRect 10 250 15 25 "Two" "plum",
96       DrawRect 300 250 15 25 "One" "plum"],
97
98     [DrawRect 300 350 20 20 "Two" "blue",
99       DrawRect 10 350 20 20 "One" "blue",
100      DrawRect 10 350 15 25 "Two" "plum",
101      DrawRect 300 350 15 25 "One" "plum"]])
102
103
104 [ Def (Viewdef "Default" (Const 400) (Const 400)), Def (Rectangle "box" (Const 10) (Const
    400) (Const 20) (Const 20) Green), Com (Move ["box"] (Abs (Const 10) (Const 200))),
    Com (Move ["box"] (Rel (Const 100) (Const 0))), Com (Move ["box"] (Abs (Const 110) (
    Const 400))), Com (Move ["box"] (Rel (Minus (Const 0) (Const 100)) (Const 0))), Def (
    Rectangle "box" (Const 60) (Const 400) (Const 20) (Const 20) Blue)]

```

B.3 *Used Hand-outs*

See listing 4.

B.4 *Sample Files*

See listings 6 and 7.

C QUESTION 3 CODE FILES

C.1 Source Code

Listing 10: src/at_server/at_server.erl

```
1  %%%-----
2  %%% @doc
3  %%% Implementation of the atomic transaction (AT) server
4  %%% @end
5  %%%-----
6  %%% Student name: Rasmus Borgsmidt
7  %%% Student KU-id: qzp823
8  %%%-----
9
10 -module(at_server).
11
12 -behavior(gen_server).
13
14 %% API exports
15 -export([
16     start/1,
17     stop/1,
18     begin_t/1,
19     doquery/2,
20     query_t/3,
21     update_t/3,
22     commit_t/2
23 ]).
24
25 %% gen_server callbacks
26 -export([
27     init/1,
28     handle_call/3,
29     handle_cast/2,
30     handle_info/2,
31     terminate/2,
32     code_change/3
33 ]).
34
35 %% Macros
36 -define(DEFAULT_TIMEOUT, 5000).
37 -define(T_REF_POS, 2).
38 -define(T_PID_POS, 3).
39
40 %% Data types
41 -record(trans, {t_ref :: reference(),
42                t_pid :: pid()
43                }).
44
45 -record(state, {user_state :: term(),
46                transactions = [] :: [ #trans{} ],
47                waiting = [] :: [ pid() ]
48                }).
49
50 %%%=====
51 %%% API
52 %%%=====
53
54 %%%-----
55 %%% @doc
56 %%% Starts a new AT server
57 %%%
58 %%% @spec start(State) -> {ok, AT}
59 %%% where
60 %%%   State = term()
61 %%%   AT = pid()
62 %%% @end
63 %%%-----
64 start(State) ->
```

```

65     gen_server:start(?MODULE, [State], []).
66
67     %%-----
68     %% @doc
69     %% Stops the specified AT server
70     %%
71     %% @spec stop(AT) -> {ok, State}
72     %% where
73     %%   AT = pid()
74     %%   State = term()
75     %% @end
76     %%-----
77     stop(AT) ->
78         gen_server:call(AT, stop).
79
80     %%-----
81     %% @doc
82     %% Runs the specified query function against the current state of the
83     %% AT server and returns the result
84     %%
85     %% The atom 'error' is returned, if the supplied query function fails
86     %%
87     %% @spec doquery(AT, Fun) -> {ok, Result} or error
88     %% where
89     %%   AT = pid()
90     %%   Fun = function(State)
91     %%   Result = term()
92     %% @end
93     %%-----
94     doquery(AT, Fun) ->
95         % We are allowing the client-provided function a 'reasonable' amount
96         % of time to complete its call, although we cannot really know how
97         % long it needs. But if we use 'infinity', we expose our AT server
98         % to the risk of being stalled indefinitely by a rogue query function
99         try
100             gen_server:call(AT, {doquery, Fun}, ?DEFAULT_TIMEOUT)
101         catch
102             _ : _ -> error
103         end.
104
105     %%-----
106     %% @doc
107     %% Begins a transaction on the current state of the AT server and returns
108     %% a transaction reference
109     %%
110     %% @spec begin_t(AT) -> {ok, Ref}
111     %% where
112     %%   AT = pid()
113     %%   Ref = reference()
114     %% @end
115     %%-----
116     begin_t(AT) ->
117         gen_server:call(AT, begin_t).
118
119     %%-----
120     %% @doc
121     %% Queries the current state of the specified transaction
122     %%
123     %% @spec query_t(AT, Ref, Fun) -> {ok, Result} or aborted
124     %% where
125     %%   AT = pid()
126     %%   Ref = reference()
127     %%   Fun = function(State)
128     %%   Result = term()
129     %% @end
130     %%-----
131     query_t(AT, Ref, Fun) ->
132         % Don't use a timeout, which can be built into the query function if necessary
133         gen_server:call(AT, {query_t, Ref, Fun}, infinity).
134
135     %%-----
136     %% @doc

```

```

137 %% Updates the current state of the specified transaction. This function is
138 %% non-blocking and returns ok immediately
139 %%
140 %% @spec update_t(AT, Ref, Fun) -> ok
141 %% where
142 %%   AT = pid()
143 %%   Ref = reference()
144 %%   Fun = function(State)
145 %% @end
146 %%-----
147 update_t(AT, Ref, Fun) ->
148     gen_server:cast(AT, {update_t, Ref, Fun}).
149
150 %%-----
151 %% @doc
152 %% Commits the specified transaction to the AT server.
153 %%
154 %% @spec commit_t(AT, Ref) -> ok / aborted
155 %% where
156 %%   AT = pid()
157 %%   Ref = reference()
158 %% @end
159 %%-----
160 commit_t(AT, Ref) ->
161     gen_server:call(AT, {commit_t, Ref}).
162
163 %%%-----
164 %%% Internal Implementation
165 %%%-----
166
167 init([UserState]) ->
168     % Trap exit messages so the AT server does not exit if a transaction
169     % process dies unexpectedly
170     process_flag(trap_exit, true),
171     {ok, #state{ user_state = UserState }}.
172
173 %%%-----
174 %%% Call-backs handling client-side requests
175 %%%-----
176
177 %%% From at_server:stop(AT) -> {ok, UserState}
178 handle_call(stop, _From, State) ->
179     {stop, normal, {ok, State#state.user_state}, State};
180
181 %%% From at_server:doquery(AT, Fun) -> {ok, Fun(UserState)} / error
182 handle_call({doquery, Fun}, _From, State) ->
183     case server_query(Fun, State) of
184         error -> {reply, error, State};
185         Result -> {reply, {ok, Result}, State}
186     end;
187
188 %%% From at_server:begin_t(AT) -> {ok, TRef}
189 handle_call(begin_t, _From, State) ->
190     {TRef, NewState} = make_trans(State),
191     {reply, {ok, TRef}, NewState};
192
193 %%% From at_server:query_t(AT, TRef, Fun) -> {ok, Fun(TransState)} / aborted
194 handle_call({query_t, TRef, Fun}, From, State) ->
195     case find_trans_ref(TRef, State) of
196         undefined -> {reply, aborted, State};
197         % Query is passed off to transaction process
198         Trans -> NewState = query_trans(Trans, Fun, From, State),
199             % Client call is blocked until query has finished
200             {noreply, NewState}
201     end;
202
203 %%% From at_server:commit_t(AT, TRef) -> ok / aborted
204 handle_call({commit_t, TRef}, _From, State) ->
205     case find_trans_ref(TRef, State) of
206         undefined -> {reply, aborted, State};
207         Trans -> NewState = commit_trans(Trans, State),
208             {reply, ok, NewState}

```

```

209     end;
210
211     %%% Default catch-all
212     handle_call(_Msg, _From, State) ->
213         {ok, State}.
214
215     %%% From at_server:(AT, TRef) -> ok (non-blocking)
216     handle_cast({update_t, TRef, Fun}, State) ->
217         case find_trans_ref(TRef, State) of
218             undefined -> {noreply, State};
219             Trans -> update_trans(Trans, Fun),
220                     {noreply, State}
221         end;
222
223     %%% Default catch-all
224     handle_cast(_Msg, State) ->
225         {noreply, State}.
226
227     %%%-----
228     %%% Call-backs handling out-of-band transaction process messages
229     %%%-----
230
231     handle_info({TPid, {query_succeeded, Result, Client}}, State) ->
232         case find_trans_pid(TPid, State) of
233             undefined ->
234                 % The query succeeded but the transaction was aborted in the meantime
235                 NewState = reply_client(Client, aborted, State),
236                 {noreply, NewState};
237             _ ->
238                 NewState = reply_client(Client, {ok, Result}, State),
239                 {noreply, NewState}
240         end;
241
242     handle_info({TPid, {query_failed, Client}}, State) ->
243         Trans = find_trans_pid(TPid, State),
244         NewState = reply_client(Client, aborted, State),
245         NewState2 = abort_trans(Trans, NewState),
246         {noreply, NewState2};
247
248     handle_info({_TPid, update_succeeded}, State) ->
249         % Deliberate no-action
250         {noreply, State};
251
252     handle_info({TPid, update_failed}, State) ->
253         Trans = find_trans_pid(TPid, State),
254         NewState = abort_trans(Trans, State),
255         {noreply, NewState};
256
257     %%% Default catch-all
258     handle_info(_Reason, State) ->
259         {noreply, State}.
260
261     %%% Default catch-all
262     terminate(_Reason, _State) ->
263         ok.
264
265     %%% Default catch-all
266     code_change(_OldVsn, State, _Extra) ->
267         {ok, State}.
268
269     %%%-----
270     %%% Utility functions
271     %%%-----
272
273     server_query(Fun, State) ->
274         try
275             Fun(State#state.user_state)
276         catch
277             _ : _ -> error
278         end.
279
280     make_trans(State) ->

```

```

281 % Link transaction process with AT server to ensure clean termination
282 % if the server is stopped with running transactions
283 {ok, TPid} = at_trans:start_link(State#state.user_state),
284 TRef = make_ref(),
285 Trans = #trans{ t_ref = TRef, t_pid = TPid },
286 NewTransactions = [Trans | State#state.transactions],
287 {TRef, State#state{ transactions = NewTransactions }}.
288
289 find_trans_ref(TRef, State) ->
290   case lists:keyfind(TRef, ?T_REF_POS, State#state.transactions) of
291     false -> undefined;
292     Trans -> Trans
293   end.
294
295 find_trans_pid(TPid, State) ->
296   case lists:keyfind(TPid, ?T_PID_POS, State#state.transactions) of
297     false -> undefined;
298     Trans -> Trans
299   end.
300
301 query_trans(Trans, Fun, Client, State) ->
302   % Query is passed off to transaction process (non-blocking)
303   % Process sends a message back when it is done
304   at_trans:doquery(Trans#trans.t_pid, Fun, Client),
305   % Add client pid to waiting list for a reply
306   Waiting = [Client | State#state.waiting],
307   State#state{ waiting = Waiting }.
308
309 update_trans(Trans, Fun) ->
310   % Update is passed off to transaction process (non-blocking)
311   % Process messages back if it fails
312   at_trans:update(Trans#trans.t_pid, Fun).
313
314 commit_trans(Trans, State) ->
315   {ok, NewUserState} = at_trans:queryall(Trans#trans.t_pid),
316   % Abort all transactions immediately and notify waiting clients
317   lists:map(fun(T) -> exit(T#trans.t_pid, abort) end, State#state.transactions),
318   lists:map(fun(C) -> gen_server:reply(C, aborted) end, State#state.waiting),
319   State#state{ user_state = NewUserState, transactions = [], waiting = [] }.
320
321 abort_trans(undefined, State) ->
322   State;
323 abort_trans(Trans, State) ->
324   exit(Trans#trans.t_pid, abort),
325   NewTransactions = lists:delete(Trans, State#state.transactions),
326   State#state{ transactions = NewTransactions }.
327
328 reply_client(Client, Msg, State) ->
329   gen_server:reply(Client, Msg),
330   NewWaiting = lists:delete(Client, State#state.waiting),
331   State#state{ waiting = NewWaiting }.

```

Listing 11: src/at_server/at_trans.erl

```

1  %%%-----
2  %%% @doc
3  %%% Implementation of the transaction process for the atomic transaction server
4  %%% @end
5  %%%-----
6  %%% Student name: Rasmus Borgsmidt
7  %%% Student KU-id: qzp823
8  %%%-----
9
10 -module(at_trans).
11
12 -behavior(gen_server).
13
14 %% API exports
15 -export([
16     start_link/1,
17     doquery/3,

```



```

18         queryall/1,
19         update/2
20     ]).
21
22 %% gen_server callbacks
23 -export([
24     init/1,
25     handle_call/3,
26     handle_cast/2,
27     handle_info/2,
28     terminate/2,
29     code_change/3]).
30
31 %% Data types
32 -record(state, { at_server :: pid(),
33                 user_state :: term()
34                 }).
35
36 %%%=====
37 %%% API
38 %%%=====
39
40 %%%-----
41 %%% @doc
42 %%% Starts a new transaction process, linking it to the AT server
43 %%%
44 %%% @spec start_link(State) -> {ok, TP}
45 %%% where
46 %%%   State = term()
47 %%%   TP = pid()
48 %%% @end
49 %%%-----
50 start_link(State) ->
51     gen_server:start_link(?MODULE, [State, self()], []).
52
53 %%%-----
54 %%% @doc
55 %%% Runs the query function against the state of the transaction but is
56 %%% non-blocking and always returns ok. The result is sent in a separate
57 %%% message later when it is ready, and the supplied token is returned
58 %%% with it
59 %%%
60 %%% @spec doquery(TP, Fun, Token) -> ok
61 %%% where
62 %%%   TP = pid()
63 %%%   Fun = function(State)
64 %%%   Token = term()
65 %%% @end
66 %%%-----
67 doquery(TP, Fun, Token) ->
68     gen_server:cast(TP, {doquery, Fun, Token}).
69
70 %%%-----
71 %%% @doc
72 %%% Returns the entire state held by this transaction process
73 %%%
74 %%% @spec queryall(TP) -> {ok, State}
75 %%% where
76 %%%   TP = pid()
77 %%%   State = term()
78 %%% @end
79 %%%-----
80 queryall(TP) ->
81     gen_server:call(TP, queryall).
82
83 %%%-----
84 %%% @doc
85 %%% Runs the update function against the state of the transaction but is
86 %%% non-blocking and always returns ok. A message is sent later to indicate
87 %%% if the operation succeeded
88 %%%
89 %%% @spec update(TP, Fun) -> ok

```

```

90 %% where
91 %%   TP = pid()
92 %%   Fun = function(State)
93 %% @end
94 %%-----
95 update(TP, Fun) ->
96     gen_server:cast(TP, {update, Fun}).
97
98 %%%-----
99 %%% Internal Implementation
100 %%%-----
101
102 %% gen_server callbacks
103
104 init([UserState, AT]) ->
105     {ok, #state{ at_server = AT, user_state = UserState}}.
106
107 %%% From at_trans:queryall(TP) -> {ok, UserState}
108 handle_call(queryall, _From, State) ->
109     {reply, {ok, State#state.user_state}, State};
110
111 %%% Default catch-all
112 handle_call(_Msg, _From, State) ->
113     {reply, ok, State}.
114
115 %%% From at_trans:doquery(TP) -> ok
116 handle_cast({doquery, Fun, Token}, State) ->
117     try Fun(State#state.user_state) of
118         Result -> tell(State#state.at_server, {query_succeeded, Result, Token}),
119                 {noreply, State}
120     catch
121         _ : _ -> tell(State#state.at_server, {query_failed, Token}),
122                 {noreply, State}
123     end;
124
125 %%% From at_trans:update(TP, Fun) -> ok
126 handle_cast({update, Fun}, State) ->
127     try Fun(State#state.user_state) of
128         NewUserState -> tell(State#state.at_server, update_succeeded),
129                         NewState = State#state{ user_state = NewUserState },
130                         {noreply, NewState}
131     catch
132         _ : _ -> tell(State#state.at_server, update_failed),
133                 {noreply, State}
134     end;
135
136 %%% Default catch-all
137 handle_cast(_Msg, State) ->
138     {noreply, State}.
139
140 %%% Default catch-all
141 handle_info(_Reason, State) ->
142     {noreply, State}.
143
144 %%% Default catch-all
145 terminate(_Reason, _State) ->
146     ok.
147
148 %%% Default catch-all
149 code_change(_OldVsn, State, _Extra) ->
150     {ok, State}.
151
152
153 %% Utility functions
154
155 tell(Recipient, Msg) ->
156     Recipient ! {self(), Msg}.

```

Listing 12: src/at_server/at_extapi.erl

```

1 %%%-----

```

```

2  %%% @doc
3  %%% Implementation of the atomic transaction server
4  %%% @end
5  %%%-----
6  %%% Student name: Rasmus Borgsmidt
7  %%% Student KU-id: qzp823
8  %%%-----
9
10 -module(at_extapi).
11
12 -export([abort/2, tryUpdate/2, ensureUpdate/2, choiceUpdate/3]).
13
14 %%%-----
15 %%% Extended API
16 %%%-----
17
18 %%%-----
19 %%% @doc
20 %%% Aborts the specified transaction
21 %%%
22 %%% @spec abort(AT, Ref) -> aborted
23 %%% where
24 %%%   AT = pid()
25 %%%   Ref = reference()
26 %%% @end
27 %%%-----
28 abort(AT, Ref) ->
29     at_server:query_t(AT, Ref, fun(_) -> throw(abort) end).
30
31 %%%-----
32 %%% @doc
33 %%% Tries to update the state on the specified server
34 %%%
35 %%% Returns: ok      if the state was updated successfully
36 %%%          error    if the supplied function causes an error
37 %%%          aborted  if another transaction is committed during the update
38 %%%
39 %%% @spec tryUpdate(AT, Fun) -> ok / error / aborted
40 %%% where
41 %%%   AT = pid()
42 %%%   Fun = function(State)
43 %%% @end
44 %%%-----
45 tryUpdate(AT, Fun) ->
46     {ok, TRef} = at_server:begin_t(AT),
47     case at_server:query_t(AT, TRef, fun(S) -> S end) of
48         aborted -> aborted;
49         {ok, State} ->
50             try Fun(State) of
51                 NewState ->
52                     ok = at_server:update_t(AT, TRef, fun(_) -> NewState end),
53                     at_server:commit_t(AT, TRef)
54             catch
55                 _ : _ -> error
56             end
57     end.
58
59 %%%-----
60 %%% @doc
61 %%% Tries to update the state on the specified server. This function will
62 %%% keep trying until it succeeds, or the supplied function causes an error
63 %%%
64 %%% Returns: ok      if the state was updated successfully
65 %%%          error    if the supplied function causes an error
66 %%%
67 %%% @spec ensureUpdate(AT, Fun) -> ok / error
68 %%% where
69 %%%   AT = pid()
70 %%%   Fun = function(State)
71 %%% @end
72 %%%-----
73 ensureUpdate(AT, Fun) ->

```

```

74     case tryUpdate(AT, Fun) of
75         aborted -> ensureUpdate(AT, Fun);
76         Result -> Result
77     end.
78
79
80 %-----
81 %% @doc
82 %% Tries to update the state on the specified server using the supplied
83 %% dyadic function and list of values. It creates a separate transaction
84 %% for each value V, and tries to set the state on the server to Fun(State, V),
85 %% where State is the current state on the server.
86 %%
87 %% Returns: {ok, NewState} if the state was updated successfully
88 %%          error          if the function failed for all the supplied values
89 %%
90 %% @spec choiceUpdate(AT, Fun, Values) -> {ok, NewState} / error
91 %% where
92 %%   AT = pid()
93 %%   Fun = function(State)
94 %%   Values = [term()]
95 %%   NewState = term()
96 %% @end
97 %-----
98 choiceUpdate(AT, Fun, Values) ->
99     ThisPid = self(),
100     lists:map(fun(V) ->
101         % Create a transaction for each value in Values
102         spawn(fun() ->
103             {ok, TP} = at_server:begin_t(AT),
104             ok = at_server:update_t(AT, TP, fun(S) -> Fun(S, V) end),
105             % Must query from the transaction itself to be sure that the
106             % result is in fact what was committed, should this transaction
107             % succeed. Just reading from the server after a successful commit
108             % is not reliable, someone else could have begun/updated/committed
109             % a transaction in the meantime
110             case at_server:query_t(AT, TP, fun(S) -> S end) of
111                 % The query will fail if other transactions have been committed
112                 aborted -> ThisPid ! {TP, aborted};
113                 {ok, Result} ->
114                     case at_server:commit_t(AT, TP) of
115                         aborted -> ThisPid ! {TP, aborted};
116                         ok -> ThisPid ! {TP, ok, Result}
117                     end
118             end
119         end)
120     end,
121     Values),
122     receive_one_ok_of(length(Values), false, result).
123
124
125 %-----
126 %%% Communication primitives
127 %-----
128
129 % Sets up a receive loop to ensure that exactly one transaction is committed
130 % successfully and get its result
131 receive_one_ok_of(0, true, Result) ->
132     {ok, Result};
133 receive_one_ok_of(0, false, _) ->
134     error;
135 receive_one_ok_of(StillToGo, GotOK, Result) ->
136     receive
137         {_TP, aborted} ->
138             receive_one_ok_of(StillToGo-1, GotOK, Result);
139         {_TP, ok, NewResult} ->
140             case GotOK of
141                 true -> error; % Multiple OKs
142                 false -> receive_one_ok_of(StillToGo-1, true, NewResult)
143             end
144     end.

```

C.2 Test Code

Listing 13: src/at_server/at_server_tests.erl

```
1 -module(at_server_tests).
2 -include_lib("eunit/include/eunit.hrl").
3
4 -define(STATE, [1,2,3,4,5]).
5 -define(REV_STATE, [5,4,3,2,1]).
6
7 %%%-----
8 %%% Tests
9 %%%-----
10
11 doquery_success_test() ->
12     AT = start(),
13     {ok, ?STATE} = at_server:doquery(AT, fun id/1),
14     stop(AT, ?STATE),
15     [].
16
17 doquery_failure_test() ->
18     AT = start(),
19     error = at_server:doquery(AT, fun fail/1),
20     stop(AT, ?STATE),
21     [].
22
23 begin_t_test() ->
24     AT = start(),
25     {ok, _} = at_server:begin_t(AT),
26     stop(AT, ?STATE),
27     [].
28
29 query_t_success_test() ->
30     AT = start(),
31     {ok, TP} = at_server:begin_t(AT),
32     {ok, ?STATE} = at_server:query_t(AT, TP, fun id/1),
33     {ok, ?STATE} = at_server:query_t(AT, TP, fun id/1),
34     stop(AT, ?STATE),
35     [].
36
37 query_t_failure_test() ->
38     AT = start(),
39     {ok, TP} = at_server:begin_t(AT),
40     aborted = at_server:query_t(AT, TP, fun fail/1),
41     aborted = at_server:query_t(AT, TP, fun id/1),
42     stop(AT, ?STATE),
43     [].
44
45 update_t_success_test() ->
46     AT = start(),
47     {ok, TP1} = at_server:begin_t(AT),
48     {ok, TP2} = at_server:begin_t(AT),
49     {ok, ?STATE} = at_server:query_t(AT, TP1, fun id/1),
50     ok = at_server:update_t(AT, TP1, fun reverse/1),
51     {ok, ?REV_STATE} = at_server:query_t(AT, TP1, fun id/1),
52     {ok, ?STATE} = at_server:doquery(AT, fun id/1),
53     {ok, ?STATE} = at_server:query_t(AT, TP2, fun id/1),
54     stop(AT, ?STATE),
55     [].
56
57 update_t_failure_test() ->
58     AT = start(),
59     {ok, TP} = at_server:begin_t(AT),
60     {ok, ?STATE} = at_server:query_t(AT, TP, fun id/1),
61     ok = at_server:update_t(AT, TP, fun fail/1),
62     aborted = at_server:query_t(AT, TP, fun id/1),
63     {ok, ?STATE} = at_server:doquery(AT, fun id/1),
64     stop(AT, ?STATE),
65     [].
66
67 commit_t_success_test() ->
```

```

68     AT = start(),
69     {ok, TP1} = at_server:begin_t(AT),
70     {ok, TP2} = at_server:begin_t(AT),
71     {ok, ?STATE} = at_server:doquery(AT, fun id/1),
72     ok = at_server:update_t(AT, TP1, fun reverse/1),
73     {ok, ?REV_STATE} = at_server:query_t(AT, TP1, fun id/1),
74     {ok, ?STATE} = at_server:query_t(AT, TP2, fun id/1),
75     ok = at_server:commit_t(AT, TP1),
76     {ok, ?REV_STATE} = at_server:doquery(AT, fun id/1),
77     aborted = at_server:query_t(AT, TP2, fun id/1),
78     stop(AT, ?REV_STATE),
79     [].
80
81 commit_t_abort_longrunning_test() ->
82     AT = start(),
83     {ok, TP1} = at_server:begin_t(AT),
84     {ok, TP2} = at_server:begin_t(AT),
85     ok = at_server:update_t(AT, TP1, fun reverse/1),
86     {ok, ?REV_STATE} = at_server:query_t(AT, TP1, fun id/1),
87     spawn(fun() -> timer:sleep(50), ok = at_server:commit_t(AT, TP1) end),
88     % When running this test, the key is that we are not waiting 30 secs
89     % for it to complete. The commit of TP1 should abort TP2 and force a return.
90     % When the line below is commented out, the test fails on a timeout
91     aborted = at_server:query_t(AT, TP2, fun wait30/1),
92     stop(AT, ?REV_STATE),
93     [].
94
95 commit_t_competing_test() ->
96     AT = start(),
97     TransCount = 100,
98     TPs = lists:map(fun(_) -> {ok, TP} = at_server:begin_t(AT), TP end,
99                     lists:seq(1, TransCount)),
100     lists:map(fun(TP) -> ok = at_server:update_t(AT, TP, fun reverse/1) end, TPs),
101     lists:map(fun(TP) -> {ok, ?REV_STATE} = at_server:query_t(AT, TP, fun id/1) end, TPs),
102     EUnitPid = self(),
103     % Attempt to commit all transactions 'simultaneously'
104     lists:map(fun(TP) -> spawn(fun() -> case at_server:commit_t(AT, TP) of
105                                     aborted -> EUnitPid ! {TP, aborted};
106                                     ok -> EUnitPid ! {TP, ok}
107                                 end
108                             end)
109             end, TPs),
110     % Depending on how many transactions are used in this test, the following
111     % call to doquery is allowed to complete with the soon-to-be outdated
112     % state. This is correct behavior, when no transaction has been fully
113     % committed yet. If a sufficient number of transactions are used (fx. 100),
114     % the time it takes to spawn the processes is enough that the server state is
115     % updated, and doquery returns the new state instead.
116     {ok, _Result} = at_server:doquery(AT, fun id/1),
117     % The following checks that exactly one transaction was committed successfully
118     ok = receive_one_ok_of(TransCount, false),
119     % When stopping the AT server, the state has always been correctly updated
120     stop(AT, ?REV_STATE),
121     [].
122
123
124 %%%-----
125 %%% Utility functions
126 %%%-----
127
128 start() ->
129     {ok, AT} = at_server:start(?STATE),
130     AT.
131
132 stop(AT, State) ->
133     {ok, State} = at_server:stop(AT),
134     ok.
135
136 id(S) -> S.
137
138 fail(_) -> throw(up).
139

```

```

140 reverse(S) -> lists:reverse(S).
141
142 wait30(S) ->
143     timer:sleep(30000),
144     S.
145
146 receive_one_ok_of(0, true) ->
147     ok;
148 receive_one_ok_of(0, false) ->
149     error;
150 receive_one_ok_of(StillToGo, GotOK) ->
151     receive
152         {_TP, aborted} ->
153             %?debugFmt("Transaction ~p aborted", [TP]),
154             receive_one_ok_of(StillToGo-1, GotOK);
155         {_TP, ok} ->
156             %?debugFmt("----> Transaction ~p committed", [TP]),
157             case GotOK of
158                 true -> error; % Multiple OKs
159                 false -> receive_one_ok_of(StillToGo-1, true)
160             end
161     end.

```

Listing 14: src/at_server/at_extapi_tests.erl

```

1 -module(at_extapi_tests).
2 -include_lib("eunit/include/eunit.hrl").
3
4 -define(STATE, [1,2,3,4,5]).
5 -define(REV_STATE, [5,4,3,2,1]).
6
7 %%%-----
8 %%% Tests
9 %%%-----
10
11 abort_test() ->
12     AT = start(),
13     {ok, TP} = at_server:begin_t(AT),
14     aborted = at_extapi:abort(AT, TP),
15     stop(AT, ?STATE),
16     [].
17
18 try_update_error_test() ->
19     AT = start(),
20     error = at_extapi:tryUpdate(AT, fun fail/1),
21     stop(AT, ?STATE),
22     [].
23
24 try_update_aborted_test() ->
25     AT = start(),
26     {ok, TP} = at_server:begin_t(AT),
27     ok = at_server:update_t(AT, TP, fun reverse/1),
28     EUnitPid = self(),
29     spawn(fun() -> EUnitPid ! at_extapi:tryUpdate(AT, fun(S) -> timer:sleep(100),
30                                                         lists:reverse(S))
31             end)
32     end),
33     timer:sleep(50),
34     ok = at_server:commit_t(AT, TP),
35     receive
36         Msg -> ?assertEqual(aborted, Msg)
37     end,
38     stop(AT, ?REV_STATE),
39     [].
40
41 try_update_ok_test() ->
42     AT = start(),
43     ok = at_extapi:tryUpdate(AT, fun reverse/1),
44     stop(AT, ?REV_STATE),
45     [].
46

```

```

47 ensure_update_error_test() ->
48     AT = start(),
49     error = at_extapi:ensureUpdate(AT, fun fail/1),
50     stop(AT, ?STATE),
51     [].
52
53 ensure_update_ok_test() ->
54     AT = start(),
55     ok = at_extapi:ensureUpdate(AT, fun reverse/1),
56     stop(AT, ?REV_STATE),
57     [].
58
59 ensure_update_retry_test() ->
60     AT = start(),
61     {ok, TP} = at_server:begin_t(AT),
62     ok = at_server:update_t(AT, TP, fun reverse/1),
63     EUnitPid = self(),
64     spawn(fun() -> EUnitPid ! at_extapi:ensureUpdate(AT, fun(S) -> timer:sleep(100),
65                                                         lists:reverse(S)
66                                                         end)
67             end),
68     timer:sleep(50),
69     ok = at_server:commit_t(AT, TP),
70     receive
71         Msg -> ?assertEqual(ok, Msg)
72     end,
73     stop(AT, ?STATE),
74     [].
75
76 choice_update_test() ->
77     AT = start(),
78     {ok, State} = at_extapi:choiceUpdate(AT, fun(S, V) ->
79                                     lists:map(fun(A) -> A+V end, S) end, [2,3,5,7,11,13]),
80     stop(AT, State),
81     [].
82
83 %%%-----
84 %%% Utility functions
85 %%%-----
86
87 start() ->
88     {ok, AT} = at_server:start(?STATE),
89     AT.
90
91 stop(AT, Expected) ->
92     {ok, State} = at_server:stop(AT),
93     ?assertEqual(Expected, State),
94     ok.
95
96 fail(_) -> throw(up).
97
98 reverse(S) -> lists:reverse(S).

```

C.3 Test Output

Listing 15: Session output: src/at_server/at_server_tests.erl

```

1 > eunit:test(at_server, [verbose]).
2 ===== EUnit =====
3 module 'at_server'
4   module 'at_server_tests'
5     at_server_tests: commit_t_competing_test...[0.007 s] ok
6     at_server_tests: commit_t_abort_longrunning_test...[0.051 s] ok
7     at_server_tests: commit_t_success_test...ok
8     at_server_tests: update_t_failure_test...ok
9     at_server_tests: update_t_success_test...ok
10    at_server_tests: query_t_failure_test...ok
11    at_server_tests: begin_t_test...ok

```



```

12     at_server_tests: query_t_success_test...ok
13     at_server_tests: doquery_failure_test...ok
14     at_server_tests: doquery_success_test...ok
15     [done in 0.087 s]
16 [done in 0.087 s]
17 =====
18 All 10 tests passed.
19 ok

```

Listing 16: Session output: src/at_server/at_extapi_tests.erl

```

1 > eunit:test(at_extapi, [verbose]).
2 ===== EUnit =====
3 module 'at_extapi'
4   module 'at_extapi_tests'
5     at_extapi_tests: choice_update_test...[0.001 s] ok
6     at_extapi_tests: ensure_update_ok_test...ok
7     at_extapi_tests: ensure_update_retry_test...[0.202 s] ok
8     at_extapi_tests: ensure_update_error_test...ok
9     at_extapi_tests: try_update_error_test...ok
10    at_extapi_tests: try_update_aborted_test...[0.101 s] ok
11    at_extapi_tests: try_update_ok_test...[0.001 s] ok
12    at_extapi_tests: abort_test...ok
13    [done in 0.328 s]
14 [done in 0.328 s]
15 =====
16 All 8 tests passed.
17 ok

```

Listing 17: Session output: commit_t_competing_test() (debug enabled)

```

1 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31695> aborted
2 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31699> aborted
3 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31704> aborted
4 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31714> aborted
5 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31719> aborted
6 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31723> aborted
7 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31728> aborted
8 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31732> aborted
9 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31736> aborted
10 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31740> aborted
11 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31744> aborted
12 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31748> aborted
13 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31758> aborted
14 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31762> aborted
15 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31768> aborted
16 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31772> aborted
17 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31776> aborted
18 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31780> aborted
19 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31784> aborted
20 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31788> aborted
21 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31790> aborted
22 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31792> aborted
23 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31794> aborted
24 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31796> aborted
25 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31800> aborted
26 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31806> aborted
27 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31810> aborted
28 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31814> aborted
29 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31818> aborted
30 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31822> aborted
31 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31826> aborted
32 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31830> aborted
33 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31832> aborted
34 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31834> aborted
35 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31836> aborted
36 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31840> aborted
37 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31844> aborted
38 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31848> aborted
39 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31852> aborted
40 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31856> aborted
41 at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31860> aborted

```

[illegible]

D SUBMITTED FILE TREE

Listing 18: File tree under src/

1 STILL TO DO