EXAM REPORT

Advanced Programming 2013

Rasmus Borgsmidt

CONTENTS

LISTINGS

LIST OF TABLES

INTRODUCTION

This report is part of an answer to the take-home exam of the course *Advanced Programming 2013* held at University of Copenhagen, Department of Computer Science (DIKU).

The intended reader is expected to have knowledge of general computer science topics equivalent to a third-year undergraduate student; and in addition to have followed this course.

The answer to this exam is submitted electronically, and this report is submitted alongside a file archive called `src.zip`; the contents of this archive is specified in listing 19.

1 Q1: SALSA LANGUAGE PARSER

In this question, we are asked to implement a parser of the SALSA source language, constructing an Abstract Syntax Tree (AST) in accordance with the data types specified in listing 4. The parser must be constructed using one of three available parser libraries, and I have chosen to use `SimpleParse` because it is simple and does what we need (listing 5).

All the relevant code and test files are included in appendix A.

1.1 *Grammar Transformations*

The supplied grammar is both ambiguous and left-recursive. But for this grammar[1] it is possible to construct an equivalent LL(1) grammar using the techniques outlined by Mogensen [1, p. 69]:

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

There is a number of benefits in doing this. First and foremost, an LL(1) grammar makes it a near-mechanical process to construct a parser using `SimpleParse` and this makes it a good choice for emphasizing correctness. The parser can be improved for brevity and efficiency later, fx. using constructs like `many1` and by combining several of the smaller parsers to minimize code repetition. Secondly as can be seen from table 1, 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 Not every grammar has an equivalent LL(1) grammar

As an aside on left-factorization, *where required* in item 3 above alludes to the fact that it is only necessary to change the grammar in this respect where *a single nonterminal* has several productions beginning with the same sequence of symbols. It is therefore not a problem that both *Pos* and *Prim* have a production beginning with `'('`.

The associativity of `'||'` was not specified by the original grammar, but I have chosen it to be left-associative since that seems most logical and is symmetric to the associativity of `'@'`. The associavity of `'||'` has no impact on the generated output anyway. `'+'` and `'-'` are assumed to have equal precedence.

The constructed parser in `SalsaParser.hs` is shown in listing 5.

1.2 *Testing*

I wanted to come up with a set of *automated tests*[2] that exercise the various parts of the grammar separately without having to expose additional functions from the module. The resulting test suite is shown in listing 2 and the session output is shown in listing 3.

1.3 *Assessment*

I believe this is a complete and correct solution to question 1, and I have a high degree of confidence in the code. I base this assessment on the fact that I was able to obtain an equivalent LL(1) grammar, which makes it straightforward to construct a parser, and that I have a reliable test suite to exercise the parser.

Furthermore, `ghc -Wall` compiles the code without any warnings.

2 automated test: test that breaks automatically if there is a problem and requires no manual supervision

Table 1: LL(1) Salsa Grammar

Program	::=	*DefComs*
DefComs	::=	*DefCom DefComs*$_*$
DefComs$_*$	::=	*DefCom DefComs*$_*$
	|	ϵ
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 Command*$_*$
Command$_*$	::=	`'||'` *Command1 Command*$_*$
	|	ϵ
Command1	::=	*Command2 Command1*$_*$
Command1$_*$	::=	`'@'` *VIdent Command1*$_*$
	|	ϵ
Command2	::=	*SIdents* `'->'` *Pos*
	|	`'{'` *Command* `'}'`
VIdents	::=	*VIdent VIdents*$_*$
VIdents$_*$	::=	*VIdent VIdents*$_*$
	|	ϵ
SIdents	::=	*SIdent SIdents*$_*$
SIdents$_*$	::=	*SIdent SIdents*$_*$
	|	ϵ
Pos	::=	`'('` *Expr* `','` *Expr* `')'`
	|	`'+' '('` *Expr* `','` *Expr* `')'`
Expr	::=	*Prim Expr*$_*$
Expr$_*$	::=	`'+'` *Prim Expr*$_*$
	|	`'-'` *Prim Expr*$_*$
	|	ϵ
Prim	::=	*integer*
	|	*SIdent* `'.'` *Coord*
	|	`'('` *Expr* `')'`
Coord	::=	`'x'` | `'y'`
Colour	::=	`'blue'` | `'plum'` | `'red'` | `'green'` | `'orange'`

2 Q2: SALSA LANGUAGE INTERPRETER

In this question, we are asked to construct an interpreter that from an AST generated by the parser constructed in question 1, generates an `Animation` in accordance with the data types specified in listing 10. The implementation is shown in listing 8 and the definitions mentioned in this section are defined there. All the relevant code and test files are included in appendix B.

A `Context` comprises an environment and state information; the environment specifies views, shapes and other definitions, the state specifies the position of each shape on the views as well as the graphics instructions generated so far by by interpreter. It is an explicit requirement that our `StateCommand` monad reflect that a command can modify only the state information, not the environment, so I have chosen these definitions:

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

The `SalsaCommand` type captures the effect of running a command in a given context, that is moving from the current key frame to the next and generating all the intermediate frames that goes with it.

The `Salsa` type represents an animation step; it is either a definition that changes the environment somehow, perhaps by modifying the current key frame, or it is a new command that causes a move to a new keyframe. To encapsulate a `SalsaCommand` as a `Salsa` computation, we use the following function:

```
1 liftC :: SalsaCommand a -> Salsa a
2 liftC sc = Salsa $ \context -> let (x, st) = runSC sc context
3                                in (x, context { state = st })
```

2.1 *Frame Sets*

To generate the frames as each command moves the animation from one context to the next, my implementation keeps track of *frame sets*; a frame set is the set of frames between one key frame and the next. A blank context starts out with a single frame in a single frame set, representing the initial key frame. With each command a new frame set is added, and when every `Salsa` computation has been applied, the final `Animation` is constructed by flattening the frame sets to a list of frames.

because `Definitions` frequently need to manipulate the current key frame, the frame sets are stored in reverse order so the current one is at the head of the list. Likewise the frames inside a single frame set are stored in reverse order, so the key frame is at the head. With a framerate of n, after m commands we have the following frame set structure where the k are key frames and f are intermediate frames:

$$[[k_{mn}, \ldots, f_{m1}], \ldots, [k_{1n}, \ldots, f_{11}], [k_0]]$$

2.2 *Parallel Commands*

To achieve the effect of running two commands in parallel, the interpreter first runs both commands as normal; then it merges the two most recent frame sets so that the generated graphics instructions occur in the same frame set. This is in fact the reason to use distinct frame sets at all; so they can be combined after the fact.

In order to do this reliably, the interpreter needs to keep track of which shapes caused the generation of which instructions. Therefore the interpreter is working with what I have called extended frames:

```
1 type ExtFrame = [(Ident, GpxInstr)]
```

This additional information is used to ensure we keep the right information from each of the merged frame sets.

2.3 *Testing*

For this question, I have resorted to manual testing by running the interpreter on input in GHCi. Sample output of this testing is show in listing 9. I have also found it helpful to verify the behavior of the interpreter by pasting generated output into the viewer supplied here: `http://www.diku.dk/~kflarsen/ap-e2013/gpx.html`.

2.4 *Assessment*

Although I have only conducted manual testing, I have been quite thorough so I have a reasonable confidence in the implementation. Furthermore, `ghc -Wall` compiles the code without any warnings. I believe this answer to be complete and correct.

However, given the stated objective of the question; "using monads for structuring your code," I wonder if there are other areas of the implementation where monads should or could have been used as well. I considered briefly creating an evaluation executor monad for the expression evaluation, but the function `eval :: Context -> Expr -> Integer` is so simple that it was unnecessary.

As an aside, I noticed that the target language has no means of tracking when views come into existence. If a view is defined later in the animation, the resulting instructions do not reflect this; the view is created immediately but stay blank until something happens on them.

3 Q3: ATOMIC TRANSACTION SERVER

In this question, we are asked to build an atomic transaction server. I have chosen to use the OTP behavior `gen_server` because it provides a nice well-structured framework for the code. All the relevant code and test files are included in appendix C.

My implementation is structured as two separate `gen_server` instances; `at_server` shown in listing 11 and `at_trans` shown in listing 12. This ensures a good separation of concerns. The `at_server` implements the AT server itself, whereas the `at_trans` implements the transaction processes that do much of the work for the server.

3.1 *Timeouts*

I considered using timeouts to protect the implementation from stalling functions, but there is really no way for the AT server implementation to determine what a "reasonable" timeout is. So I have chosen a middleground, where the `doquery` call directly to the server employs a timeout. This is the most critical call because if that hangs, the whole server is down!

For the other queries and updates, if the client wants to use a timeout this can be implemented inside the functions passed to the AT server.

3.2 *Responding to Clients*

The `update_t` call always returns `ok` immediately so this has been implemented using `handle_cast`.

The `query_t` call is handled using `handle_call` because the client is blocked waiting for a response, but the server passes off the request to the relevant transaction process and finishes the `handle_call` without replying to the client. Later when the transaction process has finished, it notifies the AT server using an out-of-band message with reason `{TPid, {query_succeeded, Result, Client}}` or `{TPid, {query_failed, Client}}` and the server responds appropriately to the client.

The `commit_t` call is also handled using `handle_call` but for this, the server blocks and responds to the client before exiting `handle_call`. This is of course to ensure that only one `commit_t` be accepted at a time.

When the transaction has been committed, the server kills all remaining transaction processes and immediately replies to all clients waiting for a reply. This is handled by a *waiting queue* that a client's pid is added to when calling into `query_t`. Using this waiting list is safe because the client is blocked when waiting for a reply, and can therefore only ever have one reply outstanding from the server.

3.3 *Extended API*

I have implemented the extended API as requested and the code can be found in listing 13. The most interesting function is `choiceUpdate` towards the end which spawns a separate process for each value in the supplied list. It then proceeds to wait for responses from the processes and ensures that it gets `ok` from exactly one process and `aborted` from the rest.

My interpretation of the exam text: "[...] the result of choiceUpdate is the result of this commit" is that `choiceUpdate` should return the resulting state that was actually committed to the server. In order to achieve this, each worker process reads back the result of applying the supplied function to the state in the transaction prior to attempting a commit. This is the only way to be sure that the correct result is returned. If `choiceUpdate` instead read the state from the server after the commit, it could easily have changed again in the meantime.

There is an interesting theoretical problem with `choiceUpdate` that I was not sure how to handle; If the supplied list is so long, that the process calling choiceUpdate is still busy spawning new processes when the first process has already finished and committed its result, then it is possible for later processes to go through a clean `begin_t/update_t/commit_t` cycle as well. Revisiting the problem now, I think a potential solution might be to create all the processes up-front and let them begin a transaction with `begin_t`. Only then should `choiceUpdate` kick off the rest of the cycle with some kind of multicast message to every worker process at once.

3.4 *Testing*

I have used EUnit for testing this implementation and the resulting automated test scripts can be found in listings 14 and 15. Output from running these can be found in listings 16 and 17. Output from a particularly interesting test can be found in listing 18, where 100 distinct processes all try to commit a transaction simultaneously. The output demonstrates how one of these was committed successfully while the 99 others were correctly aborted.

3.5 *Assessment*

I believe this is a complete and correct solution to question 3 and I have a very high degree of confidence in the implementation. I base this assessment on the fact that I have used a tried-and-true framework for structuring the code, that the implementation uses separate `gen_server` instances to ensure separation of concerns, and that I have built a reasonably comprehensive automated test suite where all the tests show that the implementation works.

BIBLIOGRAPHY

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

ACRONYMS

AST Abstract Syntax Tree

LL(1) Left-to-Right, Left-order, 1-symbol Look-ahead

A Q1 CODE FILES

A.1 *Source Code*

Listing 1: `src/salsa/SalsaParser.hs`

```
--
-- Skeleton for Salsa parser
-- To be used at the exam for Advanced Programming, B1-2013
--

module SalsaParser
    (
      Program
    , Error
    , parseString
    , parseFile
    , reserved
    ) where

import SalsaAst
import SimpleParse
import Data.Char (isLetter, isDigit, isUpper, isLower)

-- A string is used to signal an error
type Error = String

-- Reserved words
reserved :: [String]
reserved = ["viewdef", "rectangle", "circle", "view", "group",
            "blue", "plum", "red", "green", "orange"]
isReserved :: String -> Bool
isReserved w = w `elem` reserved

-- top-level parsing function that returns a program, or a string in case of failure
parseString :: String -> Either Error Program
parseString input = let res = parse (do r <- program
                                        spaces -- allows trailing whitespace in a program
                                        eof
                                        return r) input
                    in case res of
                         [] -> Left ("unable to parse input: " ++ input)
                         (r,_):_ -> Right r

-- top-level parsing function that reads its input from a file
parseFile :: FilePath -> IO (Either Error Program)
parseFile path = do input <- readFile path
                    return $ parseString input

-- Program parser
program :: Parser Program
program = defComs

-- DefComs parser
defComs :: Parser [DefCom]
defComs = do d <- defCom
             ds <- defComs_
             return (d:ds)

-- DefComs* parser
defComs_ :: Parser [DefCom]
defComs_ = (do d <- defCom
               ds <- defComs_
               return (d:ds))
           <|> return []

-- DefCom parser
defCom :: Parser DefCom
```

```
defCom = (do d <- def
             return (Def d))
         <|> (do c <- com
                 return (Com c))

-- Definition parser
def :: Parser Definition
def = (do symbol "viewdef"
          v <- vIdent
          e0 <- expr
          e1 <- expr
          return (Viewdef v e0 e1))
      <|> (do symbol "rectangle"
              s <- sIdent
              e0 <- expr
              e1 <- expr
              e2 <- expr
              e3 <- expr
              c <- col
              return (Rectangle s e0 e1 e2 e3 c))
      <|> (do symbol "circle"
              s <- sIdent
              e0 <- expr
              e1 <- expr
              e2 <- expr
              c <- col
              return (Circle s e0 e1 e2 c))
      <|> (do symbol "view"
              v <- vIdent
              return (View v))
      <|> (do symbol "group"
              v <- vIdent
              schar '['
              vs <- vIdents
              schar ']'
              return (Group v vs))

-- Command parser
com :: Parser Command
com = com1 >>= com_

-- Command* parser
com_ :: Command -> Parser Command
com_ c0 = (do symbol "||"
              c1 <- com1
              com_ (Par c0 c1))
          <|> return c0

-- Command1 parser
com1 :: Parser Command
com1 = com2 >>= com1_

-- Command1* parser
com1_ :: Command -> Parser Command
com1_ c = (do schar '@'
              v <- vIdent
              com1_ (At c v))
          <|> return c

-- Command2 parser
com2 :: Parser Command
com2 = (do ss <- sIdents
           symbol "->"
           p <- pos
           return (Move ss p))
       <|> (do schar '{'
               c <- com
               schar '}'
               return c)

-- VIdents parser
vIdents :: Parser [Ident]
```

```
vIdents = do v <- vIdent
             vs <- vIdents_
             return (v:vs)

-- VIdents* parser
vIdents_ :: Parser [Ident]
vIdents_ = (do many1 space -- identifiers must be separated by whitespace
               v <- vIdent
               vs <- vIdents_
               return (v:vs))
           <|> return []

-- SIdents parser
sIdents :: Parser [Ident]
sIdents = do s <- sIdent
             ss <- sIdents_
             return (s:ss)

-- SIdents* parser
sIdents_ :: Parser [Ident]
sIdents_ = (do many1 space -- identifiers must be separated by whitespace
               s <- sIdent
               ss <- sIdents_
               return (s:ss))
           <|> return []

-- Pos parser
pos :: Parser Pos
pos = (do schar '('
          e0 <- expr
          schar ','
          e1 <- expr
          schar ')'
          return (Abs e0 e1))
      <|> (do schar '+'
              schar '('
              e0 <- expr
              schar ','
              e1 <- expr
              schar ')'
              return (Rel e0 e1))

-- Expr parser
expr :: Parser Expr
expr = prim >>= expr_

-- Expr* parser
expr_ :: Expr -> Parser Expr
expr_ e0 = (do schar '+'
               e1 <- prim
               expr_ (Plus e0 e1))
          <|> (do schar '-'
                  e1 <- prim
                  expr_ (Minus e0 e1))
          <|> return e0

-- Prim parser
prim :: Parser Expr
prim = (do i <- integer
           return (Const i))
       <|> (do s <- sIdent
               schar '.'
               proj s)
       <|> (do schar '('
               e <- expr
               schar ')'
               return e)

-- This parser function handles the coordinate selection in Prim expressions
proj :: Ident -> Parser Expr
proj s = (do schar 'x'
             return (Xproj s))
```

```
         <|> (do schar 'y'
                 return (Yproj s))

-- Colour parser
col :: Parser Colour
col = (symbol "blue" >> return Blue)
      <|> (symbol "plum" >> return Plum)
      <|> (symbol "red" >> return Red)
      <|> (symbol "green" >> return Green)
      <|> (symbol "orange" >> return Orange)

-- integers
integer :: Parser Integer
integer = token (do intstr <- many1 $ satisfy isDigit
                    return (read intstr))

-- identifiers
vIdent :: Parser Ident
vIdent = ident isUpper

sIdent :: Parser Ident
sIdent = ident isLower

ident :: (Char -> Bool) -> Parser Ident
ident leading = token (do c <- satisfy leading
                          cs <- letdigs
                          if (c:cs) `elem` reserved
                          then reject
                          else return (c:cs))
    where letter = satisfy isLetter
          digit = satisfy isDigit
          letdigs = many (letter <|> digit <|> char '_')
```

A.2 *Test Code*

Listing 2: `src/salsa/SalsaParserTest.hs`

```
import Control.Exception (assert)
import SalsaParser
import SalsaAst

-- parses a string and compares the AST to an expected program
run :: String -> Program -> String
run s p = assert (Right p == parseString s) (shw "pass" s)

-- parses a string and checks that an error occurs
err :: String -> Program -> String
err s _ = assert (isError $ parseString s) (shw "error (expected)" s)

-- indicates if a given result is an error
isError :: Either String Program -> Bool
isError (Left _) = True
isError (Right _) = False

-- formats test output
shw :: String -> String -> String
shw pre "" = pre ++ ": " ++ "<empty string>"
shw pre s = pre ++ ": " ++ s

-- main runs the test suite
main = do putStrLn "\n*** Checking Colour ***"
          putStrLn $ checkCol run "blue" Blue
          putStrLn $ checkCol run "plum" Plum
          putStrLn $ checkCol run "red" Red
          putStrLn $ checkCol run "green" Green
          putStrLn $ checkCol run "orange" Orange
          putStrLn $ checkCol err "violet" Red
          putStrLn $ checkCol err "greeen" Green
          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 SIdents ***"
64          putStrLn $ checkSIdents run "a" ["a"]
65          putStrLn $ checkSIdents run "aBC" ["aBC"]
66          putStrLn $ checkSIdents run "a12T" ["a12T"]
67          putStrLn $ checkSIdents run "aa bb cc" ["aa", "bb", "cc"]
68          putStrLn $ checkSIdents err "1abc" []
69          putStrLn $ checkSIdents err "Abc" []
70          putStrLn $ checkSIdents err "_abc" []
71          putStrLn $ checkSIdents err "_" []
72          mapM_ putStrLn $ (map $ flip (checkSIdents err) []) reserved
73
74          putStrLn "\n*** Checking VIdents ***"
75          putStrLn $ checkVIdents run "A" ["A"]
76          putStrLn $ checkVIdents run "Abc" ["Abc"]
77          putStrLn $ checkVIdents run "A12t" ["A12t"]
78          putStrLn $ checkVIdents run "AA BB CC" ["AA", "BB", "CC"]
79          putStrLn $ checkVIdents err "1Abc" []
80          putStrLn $ checkVIdents err "aBC" []
81          putStrLn $ checkVIdents err "_ABC" []
82          putStrLn $ checkVIdents 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)}}}"
```

```
                    (Move ["a"] (Abs (Const 0) (Const 0)))

        putStrLn "\n*** Checking Definition ***"
        putStrLn $ checkDefinition run "viewdef V 0 0" (Viewdef "V" (Const 0) (Const 0))
        putStrLn $ checkDefinition run "rectangle r 0 0 0 0 blue"
                    (Rectangle "r" (Const 0) (Const 0) (Const 0) (Const 0) Blue)
        putStrLn $ checkDefinition run "circle c 0 0 0 blue"
                    (Circle "c" (Const 0) (Const 0) (Const 0) Blue)
        putStrLn $ checkDefinition run "view V" (View "V")
        putStrLn $ checkDefinition run "group G [X Y Z]" (Group "G" ["X", "Y", "Z"])
        putStrLn $ checkDefinition err "view1 V" (View "V")

        putStrLn "\n*** Checking Program ***"
        putStrLn $ err "" []

        putStrLn $ run ("viewdef Default 400 400\n" ++
                        "rectangle box 10 400 20 20 green\n" ++
                        "box -> (10, 200)\n" ++
                        "box -> +(100, 0)\n" ++
                        "box -> (110,400)\n" ++
                        "box -> +(0-100, 0)\n")

                    [ 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)))]

        putStrLn $ run ("viewdef One 500 500\n" ++
                        "viewdef Two 400 400\n" ++
                        "group Both [One Two]\n" ++
                        "view Both\n" ++
                        "rectangle larry 10 350 20 20 blue\n" ++
                        "rectangle fawn 300 350 15 25 plum\n\n" ++
                        "view Two\n" ++
                        "larry -> (300, 350) || fawn -> (10,350)\n\n" ++
                        "view Both\n" ++
                        "larry fawn -> +(0, 0 - 300)")

                    [ 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))))]

        putStrLn "\n*** Checking parseFile ***"
        Right ast <- parseFile "multi.salsa"
        contents <- readFile  "multi.salsa"
        putStrLn $ flip assert (shw "pass" contents) (ast ==
                    [ 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))))])

          putStrLn "\n*** All tests completed successfully ***\n"


    where
      checkCol f s c = f ("circle c 0 0 0 " ++ s)
                         [Def (Circle "c" (Const 0) (Const 0) (Const 0) c)]
      checkExpr f s e = f ("a -> (" ++ s ++ ", " ++ s ++ ")") [Com (Move ["a"] (Abs e e))]
      checkPos f s p = f ("a -> " ++ s) [Com (Move ["a"] p)]
      checkSIdents f s ids = f (s ++ " -> (0, 0)")
                               [Com (Move ids (Abs (Const 0) (Const 0)))]
      checkVIdents f s ids = f ("group V [" ++ s ++ "]") [Def (Group "V" ids)]
      checkCommand f s c = f s [Com c]
      checkDefinition f s d = f s [Def d]
```

A.3 *Test Output*

Listing 3: Session output: `src/salsa/SalsaParserTest`

```
*** Checking Colour ***
pass: circle c 0 0 0 blue
pass: circle c 0 0 0 plum
pass: circle c 0 0 0 red
pass: circle c 0 0 0 green
pass: circle c 0 0 0 orange
error (expected): circle c 0 0 0 violet
error (expected): circle c 0 0 0 greeen
error (expected): circle c 0 0 0 Blue
error (expected): circle c 0 0 0 blue1

*** Checking Prim ***
pass: a -> (0, 0)
pass: a -> (42, 42)
pass: a -> (999999999, 999999999)
pass: a -> ((((42))), (((42))))
pass: a -> (john . x, john . x)
pass: a -> (john . y, john . y)
error (expected): a -> (-5, -5)
error (expected): a -> (-5, -5)
error (expected): a -> (42.2, 42.2)
error (expected): a -> (.8, .8)

*** Checking Expr ***
pass: a -> (1 + 2, 1 + 2)
pass: a -> (1 - 2, 1 - 2)
pass: a -> (1 + 2 + 3, 1 + 2 + 3)
pass: a -> (1 - 2 - 3, 1 - 2 - 3)
pass: a -> (1 + 2 - 3, 1 + 2 - 3)
pass: a -> (1 - 2 + 3, 1 - 2 + 3)

*** Checking Pos ***
pass: a -> (0, 0)
pass: a -> + (0, 0)

*** Checking SIdents ***
pass: a -> (0, 0)
pass: aBC -> (0, 0)
pass: a12T -> (0, 0)
pass: aa bb cc -> (0, 0)
error (expected): 1abc -> (0, 0)
error (expected): Abc -> (0, 0)
error (expected): _abc -> (0, 0)
```

```
error (expected): _ -> (0, 0)
error (expected): viewdef -> (0, 0)
error (expected): rectangle -> (0, 0)
error (expected): circle -> (0, 0)
error (expected): view -> (0, 0)
error (expected): group -> (0, 0)
error (expected): blue -> (0, 0)
error (expected): plum -> (0, 0)
error (expected): red -> (0, 0)
error (expected): green -> (0, 0)
error (expected): orange -> (0, 0)

*** Checking VIdents ***
pass: group V [A]
pass: group V [Abc]
pass: group V [A12t]
pass: group V [AA BB CC]
error (expected): group V [1Abc]
error (expected): group V [aBC]
error (expected): group V [_ABC]
error (expected): group V [_]

*** Checking Command ***
pass: a->(0, 0)
pass: a->(0, 0)@V
pass: a->(0, 0)@V@W
pass: a->(0, 0)||b->(0, 0)
pass: a->(0, 0)||b->(0, 0)||c->(0,0)
pass: a->(0, 0)||b->(0, 0)@V
pass: {a->(0, 0)||b->(0, 0)}@V
pass: {{{a->(0, 0)}}}

*** Checking Definition ***
pass: viewdef V 0 0
pass: rectangle r 0 0 0 0 blue
pass: circle c 0 0 0 blue
pass: view V
pass: group G [X Y Z]
error (expected): view1 V

*** Checking Program ***
error (expected): <empty string>
pass: viewdef Default 400 400
rectangle box 10 400 20 20 green
box -> (10, 200)
box -> +(100, 0)
box -> (110,400)
box -> +(0-100, 0)

pass: viewdef One 500 500
viewdef Two 400 400
group Both [One Two]
view Both
rectangle larry 10 350 20 20 blue
rectangle fawn 300 350 15 25 plum

view Two
larry -> (300, 350) || fawn -> (10,350)

view Both
larry fawn -> +(0, 0 - 300)

*** Checking parseFile ***
pass: viewdef One 500 500
viewdef Two 400 400
```

```
group Both [One Two]
view Both
rectangle larry 10 350 20 20 blue
rectangle fawn 300 350 15 25 plum

view Two
larry -> (300, 350) || fawn -> (10,350)

view Both
larry fawn -> +(0, 0 - 300)

*** All tests completed successfully ***
```

A.4 *Used Hand-outs*

Listing 4: `src/salsa/SalsaAst.hs`

```
module SalsaAst where

type Program = [DefCom]
data DefCom = Def Definition
            | Com Command
            deriving (Show, Eq)
data Definition = Viewdef Ident Expr Expr
                | Rectangle Ident Expr Expr Expr Expr Colour
                | Circle Ident Expr Expr Expr Colour
                | View Ident
                | Group Ident [Ident]
                deriving (Show, Eq)
data Command = Move [Ident] Pos
             | At Command Ident
             | Par Command Command
             deriving (Show, Eq)
data Pos = Abs Expr Expr
         | Rel Expr Expr
         deriving (Show, Eq)
data Expr = Plus Expr Expr
          | Minus Expr Expr
          | Const Integer
          | Xproj Ident
          | Yproj Ident
          deriving (Show, Eq)
data Colour = Blue | Plum | Red | Green | Orange
            deriving (Show, Eq)
type Ident = String
```

Listing 5: `src/salsa/SimpleParse.hs`

```
{-
  Example code from Advanced Programming lecture.

  Small monadic parser combinator library.

  Date: Sep 20, 2012
  Author: Ken Friis Larsen <kflarsen@diku.dk>
-}
module SimpleParse where

import Control.Monad(MonadPlus(..))
import Data.Char (isSpace)

newtype Parser a = Parser (String -> [(a, String)])
parse (Parser p) = p

parse' p s = [ result | (result,rest) <- parse p s, null rest ]

```

```
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
```

```
             return (v:vs)

sepBy           :: Parser a -> Parser b -> Parser [a]
p `sepBy` sep    = (p `sepBy1` sep) <|> return []

sepBy1          :: Parser a -> Parser b -> Parser [a]
p `sepBy1` sep   = do {a <- p; as <- many (do {sep; p}); return (a:as)}

chainl          :: Parser a -> Parser (a -> a -> a) -> a -> Parser a
chainl p op a    = (p `chainl1` op) <|> return a

chainl1         :: Parser a -> Parser (a -> a -> a) -> Parser a
p `chainl1` op   = do a <- p
                      rest a
                   where
                      rest a = do f <- op
                                  b <- p
                                  rest (f a b)
                               <|> return a




option :: Parser a -> Parser (Maybe a)
option p = do v <- p
              return (Just v)
           <|> return Nothing


-- Lexical combinators: ---------------------------------------------

space           :: Parser Char
space            = satisfy isSpace

spaces          :: Parser String
spaces           = many space

token           :: Parser a -> Parser a
token p          = spaces >> p

symbol           = token . string
schar            = token . char
```

A.5 *Sample Files*

Listing 6: `src/salsa/simple.salsa`

```
viewdef Default 400 400
rectangle box 10 400 20 20 green
box -> (10, 200)
box -> +(100, 0)
box -> (110,400)
box -> +(0-100, 0)
```

Listing 7: `src/salsa/multi.salsa`

```
viewdef One 500 500
viewdef Two 400 400
group Both [One Two]
view Both
rectangle larry 10 350 20 20 blue
rectangle fawn 300 350 15 25 plum

view Two
larry -> (300, 350) || fawn -> (10,350)

view Both
```

```
larry fawn -> +(0, 0 - 300)
```

B Q2 CODE FILES

B.1 *Source Code*

Listing 8: `src/salsa/SalsaInterp.hs`

```
--
-- Skeleton for Salsa interpreter
-- To be used at the exam for Advanced Programming, B1-2013
--

module SalsaInterp (Position, interpolate, runProg)
where

import SalsaAst
import Gpx
import Data.List(intersect, union, (\\))
import qualified Data.Map as M

type Position = (Integer, Integer)


--
-- Primary top-level function
--

runProg :: Integer -> Program -> Animation
runProg framerate program =
    let salsas = map defCom program
        ((), context) = runSalsa (sequence_ salsas) $ baseContext framerate
    in animation context

--
-- The function interpolate
--

interpolate :: Integer -> Position -> Position -> [Position]
interpolate rate (x0, y0) (x1, y1)
    | rate <= 0 = error "framerate must be positive"
    | otherwise = zip (ipol rate x0 x1) (ipol rate y0 y1)

ipol :: Integer -> Integer -> Integer -> [Integer]
ipol rate start end =
    let step = (fromIntegral $ end - start) / fromIntegral rate
    in map ((start +) . round . (* step)) ([1.0..fromIntegral rate] :: [Double])

--
-- Data type representing the two types of shape
--
-- A shape has an identifier, some shape-specific information, a color name
-- and a list of the views it is defined on
data Shape = Rect Ident Integer Integer String [ViewT]
           | Circ Ident Integer String [ViewT]
  deriving (Show, Eq, Ord)

idnt :: Shape -> Ident
idnt (Rect ident _ _ _ _) = ident
idnt (Circ ident _ _ _) = ident

vs :: Shape -> [ViewT]
vs (Rect _ _ _ _ vws) = vws
vs (Circ _ _ _ vws) = vws

--
-- Data types relating to the (by commands) read-only environment and writable state
--

-- 'T' suffix is just to distinguish from Salsa AST constructors
type ViewT = (Ident, Integer, Integer)
type GroupT = (Ident, [ViewT])
```

```
type ViewMap = M.Map Ident ViewT
type ShapeMap = M.Map Ident Shape
type GroupMap = M.Map Ident GroupT
-- The Context represents all the information, both read-only and writable
data Context = Context {
      views :: ViewMap
    , shapes :: ShapeMap
    , groups :: GroupMap
    , activeViews :: [ViewT]
    , frameRate :: Integer
    , state :: State
    }  deriving (Show, Eq)

type PosMap = M.Map Ident (M.Map ViewT Position)
-- An ExtFrame associates each graphics command with the shape it draws
-- This information is used to combine frame sets produced by commands that need
-- to run in parallel
type ExtFrame = [(Ident, GpxInstr)]
type FrameSet = [ExtFrame]
-- The State data type is used for the writable information
-- shapePos is a map of maps, storing the current position of each shape on each view
-- A FrameSet stores 'framerate' number of frames, and each frame set holds the
-- frames between one key frame (excl) and the next (incl)
-- The frames of a frame set are stored in reverse order, so the next key frame
-- of a frame set is always at the head of the list
-- The frame sets are also held in reverse order with the most recent at the head
data State = State {
      shapePos :: PosMap
    , frameSets :: [FrameSet]
    } deriving (Show, Eq)

blankFrame :: ExtFrame
blankFrame = []

-- The base state has a single frame in a single frame set representing the single
-- initial key frame
baseState :: State
baseState = State M.empty [[blankFrame]]

-- The base context holds empty maps, the framerate and a base state
baseContext :: Integer -> Context
baseContext rate = Context M.empty M.empty M.empty [] rate baseState

--
-- functions to query and manipulate the context
--

updateViews :: (ViewMap -> ViewMap) -> Context -> Context
updateViews f = \context -> context { views = f (views context) }

updateShapes :: (ShapeMap -> ShapeMap) -> Context -> Context
updateShapes f = \context -> context { shapes = f (shapes context) }

updateGroups :: (GroupMap -> GroupMap) -> Context -> Context
updateGroups f = \context -> context { groups = f (groups context) }

updateState :: (State -> State) -> Context -> Context
updateState f = \context -> context { state = f (state context) }

updateFrameSets :: ([FrameSet] -> [FrameSet]) -> State -> State
updateFrameSets f = \st -> st { frameSets = f (frameSets st) }

updateShapePos :: (PosMap -> PosMap) -> State -> State
updateShapePos f = \st -> st { shapePos = f (shapePos st) }

-- the key frame of a given context is always the latest frame in the latest frame set
-- and a context always has at least a key frame
updateKeyFrame :: (ExtFrame -> ExtFrame) -> [FrameSet] -> [FrameSet]
updateKeyFrame f = \((kf:set):sets) -> ((f kf):set):sets

-- Returns the shapes present on the active views
activeShapes :: Context -> [Shape]
```

```
activeShapes context = let shps = M.elems $ shapes context
                           vws = activeViews context
                       in filter (any (`elem` vws) . vs) shps

-- Helper function that determines which shapes need to move
shapesToMove :: [Ident] -> Context -> [Shape]
shapesToMove idents context =
    let moving = lookupObjs "moveShapes" context idents (shapes context)
        active = activeShapes context
    in if null $ moving \\ active
       then moving
       else error $ "shapes not defined on all active views: " ++ show idents
                ++ "\ncontext: " ++ show context

-- Helper function that returns the position of a given shape on a given view
shapeViewPos :: Shape -> ViewT -> Context -> Position
shapeViewPos shape view context =
    let posMap = shapePos $ state context
        pmap = lookupObj "" context (idnt shape) posMap
    in lookupObj "" context view pmap

activePos :: Shape -> Context -> [Position]
activePos shape context =
    let vws = activeViews context
        pmap = lookupObj "activePos" context (idnt shape) $ shapePos $ state context
    in lookupObjs "activePos" context vws pmap

updateActivePos :: Maybe Pos -> Shape -> Context -> Context
updateActivePos pos shape context =
    updateState (updateShapePos $ updatePosMap pos shape context) context

updatePosMap :: Maybe Pos -> Shape -> Context -> PosMap -> PosMap
updatePosMap pos shape context =
    \pmap -> let fromPos = activePos shape context
                 toPos = getToPositions context fromPos pos
                 vws = activeViews context
                 mp = lookupObj "updatePosMap" context (idnt shape) pmap
                 mp' = foldl (\m (k, v) -> M.insert k v m) mp $ zip vws toPos
             in M.insert (idnt shape) mp' pmap

addToPosMap :: Ident -> Position -> Context -> PosMap -> PosMap
addToPosMap ident pos context =
    \pmap -> let vws = activeViews context
                 mp = foldl (\m k -> M.insert k pos m) M.empty vws
             in M.insert ident mp pmap

-- Returns the Animation produced by this interpreter
-- This just flattens the list of frames and reverses it
animation :: Context -> Animation
animation context =
    let vws = M.elems $ views context
        fs = reverse . concat . frameSets $ state context
    in (vws , map stripFrame fs)

-- Removes the additional shape information from each frame to produce frames
-- suitable for the graphics backend
stripFrame :: ExtFrame -> Frame
stripFrame pairs = snd $ unzip pairs


--
-- Functions generating graphics instructions for moving shapes
--

-- Top-level function that causes the generation of the necessary graphics
-- instructions to move a set of shapes to a new absolute or relative position.
-- This function is responsible for generating each new frame set.
-- Strategy: Generate a complete set of instructions to draw everything in place;
-- then overwrite the instructions for the shapes that needs to move
moveShapes :: [Ident] -> Pos -> Context -> State
moveShapes idents pos context =
    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
```

```
 - the occurrence of a single Nothing forces the combined result to Nothing -}
    case M.lookup ident (views context) of
      Just view -> context { activeViews = [view] }
      Nothing -> case M.lookup ident (groups context) of
                   Just (_, vws) -> context { activeViews = vws }
                   Nothing -> error $ "undefined view or group: " ++ show ident


--
-- Monad types SalsaCommand and Salsa
--

-- 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, that is a move
-- from the current key frame to a new
newtype SalsaCommand a = SalsaCommand { runSC :: Context -> (a, State) }
instance Monad SalsaCommand where
    return x = SalsaCommand $ \context -> (x, state context)
    m >>= f = SalsaCommand $ \context -> let (x, st) = runSC m context
                                         in runSC (f x) context { state = st }

-- The Salsa type represents an animation step; it is either a definition
-- activating something or adding a new shape to views on the current key frame,
-- or it is a new command that causes the generation of a new frame set from one
-- key frame to the next
newtype Salsa a = Salsa { runSalsa :: Context -> (a, Context) }
instance Monad Salsa where
    return x = Salsa $ \context -> (x, context)
    m >>= f = Salsa $ \context -> let (x, context') = runSalsa m context
                                  in runSalsa (f x) context'

-- Changes the context locally for the command
local :: (Context -> Context) -> SalsaCommand a -> SalsaCommand a
local f m = SalsaCommand $ \context -> runSC m (f context)

-- Captures the effect of a command in a SalsaCommand
command :: Command -> SalsaCommand ()
command (At com ident) = local (activate ident) $ command com
command (Par com0 com1) = command com0 >> command com1 >> mergeConcurrent com0 com1 ()
command (Move idents pos) =
    SalsaCommand $ \context -> ((), moveShapes idents pos context)

-- Helper function that captures the effect of running to commands in parallel
mergeConcurrent :: Command -> Command -> a -> SalsaCommand a
mergeConcurrent com0 com1 x =
    SalsaCommand $ \context ->
        (x, updateFrameSets (mergeFrameSets com0 com1) $ state context)

-- Helper function that merges the two latest frame sets. This is necessary when
-- commands should run in parallel and therefore manipulate the same frame set
-- Strategy: take the latest frame set (head of list), which was generated by
-- com1, and copy across any instruction pertaining to shapes manipulated by com0.
mergeFrameSets :: Command -> Command -> [FrameSet] -> [FrameSet]
mergeFrameSets com0 com1 (fs1:fs0:sets) =
    let shps0 = shapesFromCommand com0
        shps1 = shapesFromCommand com1
    in if null $ intersect shps0 shps1
       then (zipWith (mergeFrames shps1 shps0) fs1 fs0):sets
       else error $ "concurrent commands manipulating same shapes: "
                 ++ show shps0 ++ " and " ++ show shps1
mergeFrameSets _ _ _ = error "invalid frame set configuration"

-- Strategy: In order not to overwrite the wrong instructions, we take the two
-- frame sets and remove from each, any instruction pertaining to a shape
-- manipulated ny the other command. After this, it is safe just to combine the
-- frame sets with union
mergeFrames :: [Ident] -> [Ident] -> ExtFrame -> ExtFrame -> ExtFrame
mergeFrames shps1 shps0 frame1 frame0 =
    let frame0' = filter (\(ident, _) -> not $ ident `elem` shps1) frame0
        frame1' = filter (\(ident, _) -> not $ ident `elem` shps0) frame1
    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 idents _) = idents
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 idents) =
399     Salsa $ \context -> ((), addGroup ident idents 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 idents m = map (flip (lookupObj msg context) m) idents
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"
```

```
colorName Plum = "plum"
colorName Red = "red"
colorName Green = "green"
colorName Orange = "orange"
```

B.2 *Test Output*

Listing 9: Test output for manual interpreter testing (pretty printed)

```
--
-- Interpolate tests
--

   interpolate 1 (0,0) (100,100)
=> [(100,100)]

   interpolate 2 (0,0) (100,100)
=> [(50,50),(100,100)]

   interpolate 5 (0,0) (100,100)
=> [(20,20),(40,40),(60,60),(80,80),(100,100)]


--
-- Testing empty input
--

   runProg 1 []
=> ([],[[]])

   runProg 10 []
=> ([],[[]])


--
-- Testing generation of key frames by using framerate 1
--

   runProg 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))) ]
=> ([("Default",400,400)],
    [[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"],
     [DrawRect 10 400 20 20 "Default" "green"]])

   runProg 1 [ 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)))) ]
=> ([("One",500,500),("Two",400,400)],
    [[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"],

     [DrawRect 10 350 15 25 "Two" "plum",
      DrawRect 300 350 15 25 "One" "plum",
      DrawRect 300 350 20 20 "Two" "blue",
      DrawRect 10 350 20 20 "One" "blue"],

     [DrawRect 300 50 20 20 "Two" "blue",
      DrawRect 10 50 20 20 "One" "blue",
      DrawRect 10 50 15 25 "Two" "plum",
      DrawRect 300 50 15 25 "One" "plum"]])


--
-- Testing full operation with a higher framerate that requires intermediate
-- frames (highlighted with extra indentation)
--

   runProg 3 [ 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)))) ]
=> ([("One",500,500),("Two",400,400)],
    [[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"],

         [DrawRect 203 350 15 25 "Two" "plum",
          DrawRect 300 350 15 25 "One" "plum",
          DrawRect 107 350 20 20 "Two" "blue",
          DrawRect 10 350 20 20 "One" "blue"],

         [DrawRect 107 350 15 25 "Two" "plum",
          DrawRect 300 350 15 25 "One" "plum",
          DrawRect 203 350 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",
      DrawRect 300 350 20 20 "Two" "blue",
      DrawRect 10 350 20 20 "One" "blue"],

         [DrawRect 300 250 20 20 "Two" "blue",
          DrawRect 10 250 20 20 "One" "blue",
          DrawRect 10 250 15 25 "Two" "plum",
          DrawRect 300 250 15 25 "One" "plum"],

```

```
          [DrawRect 300 150 20 20 "Two" "blue",
           DrawRect 10 150 20 20 "One" "blue",
           DrawRect 10 150 15 25 "Two" "plum",
           DrawRect 300 150 15 25 "One" "plum"],

     [DrawRect 300 50 20 20 "Two" "blue",
      DrawRect 10 50 20 20 "One" "blue",
      DrawRect 10 50 15 25 "Two" "plum",
      DrawRect 300 50 15 25 "One" "plum"]])
```

B.3 *Used Hand-outs*

The source language of the interpreter is the same as the target language of the parser and is specified in listing 4. The target language of the interpreter is specified in listing 10 below.

Listing 10: `src/salsa/Gpx.hs`

```
module Gpx where

type ViewName = String
type ColourName = String
type Frame = [GpxInstr]
type Animation = ([(ViewName, Integer, Integer)], [Frame])
data GpxInstr = DrawRect Integer Integer Integer Integer ViewName ColourName
              | DrawCirc Integer Integer Integer ViewName ColourName
              deriving (Eq, Show)
```

C Q3 CODE FILES

C.1 *Source Code*

Listing 11: `src/at_server/at_server.erl`

```
%%%-------------------------------------------------------------------
%%% @doc
%%% Implementation of the atomic transaction (AT) server
%%% @end
%%%-------------------------------------------------------------------
%%% Student name: Rasmus Borgsmidt
%%% Student KU-id: qzp823
%%%-------------------------------------------------------------------

-module(at_server).

-behavior(gen_server).

%% API exports
-export([
         start/1,
         stop/1,
         begin_t/1,
         doquery/2,
         query_t/3,
         update_t/3,
         commit_t/2
        ]).

%% gen_server callbacks
-export([
         init/1,
         handle_call/3,
         handle_cast/2,
         handle_info/2,
         terminate/2,
         code_change/3
        ]).

%% Macros
-define(DEFAULT_TIMEOUT, 5000).
-define(T_REF_POS, 2).
-define(T_PID_POS, 3).

%% Data types
-record(trans, {t_ref :: reference(),
                t_pid :: pid()
               }).

-record(state, {user_state :: term(),
                transactions = [] :: [ #trans{} ],
                waiting = [] :: [ pid() ]
               }).

%%%===================================================================
%%% API
%%%===================================================================

%%--------------------------------------------------------------------
%% @doc
%% Starts a new AT server
%%
%% @spec start(State) -> {ok, AT}
%% where
%%   State = term()
%%   AT = pid()
%% @end
%%--------------------------------------------------------------------
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) ->
```

```
    % Link transaction process with AT server to ensure clean termination
    % if the server is stopped with running transactions
    {ok, TPid} = at_trans:start_link(State#state.user_state),
    TRef = make_ref(),
    Trans = #trans{ t_ref = TRef, t_pid = TPid },
    NewTransactions = [Trans | State#state.transactions],
    {TRef, State#state{ transactions = NewTransactions }}.

find_trans_ref(TRef, State) ->
    case lists:keyfind(TRef, ?T_REF_POS, State#state.transactions) of
        false -> undefined;
        Trans -> Trans
    end.

find_trans_pid(TPid, State) ->
    case lists:keyfind(TPid, ?T_PID_POS, State#state.transactions) of
        false -> undefined;
        Trans -> Trans
    end.

query_trans(Trans, Fun, Client, State) ->
    % Query is passed off to transaction process (non-blocking)
    % Process sends a message back when it is done
    at_trans:doquery(Trans#trans.t_pid, Fun, Client),
    % Add client pid to waiting list for a reply
    Waiting = [Client | State#state.waiting],
    State#state{ waiting = Waiting }.

update_trans(Trans, Fun) ->
    % Update is passed off to transaction process (non-blocking)
    % Process messages back if it fails
    at_trans:update(Trans#trans.t_pid, Fun).

commit_trans(Trans, State) ->
    {ok, NewUserState} = at_trans:queryall(Trans#trans.t_pid),
    % Abort all transactions immediately and notify waiting clients
    lists:map(fun(T) -> exit(T#trans.t_pid, abort) end, State#state.transactions),
    lists:map(fun(C) -> gen_server:reply(C, aborted) end, State#state.waiting),
    State#state{ user_state = NewUserState, transactions = [], waiting = [] }.

abort_trans(undefined, State) ->
    State;
abort_trans(Trans, State) ->
    exit(Trans#trans.t_pid, abort),
    NewTransactions = lists:delete(Trans, State#state.transactions),
    State#state{ transactions = NewTransactions }.

reply_client(Client, Msg, State) ->
    gen_server:reply(Client, Msg),
    NewWaiting = lists:delete(Client, State#state.waiting),
    State#state{ waiting = NewWaiting }.
```

Listing 12: `src/at_server/at_trans.erl`

```
%%%-------------------------------------------------------------------
%%% @doc
%%% Implementation of the transaction process for the atomic transaction server
%%% @end
%%%-------------------------------------------------------------------
%%% Student name: Rasmus Borgsmidt
%%% Student KU-id: qzp823
%%%-------------------------------------------------------------------

-module(at_trans).

-behavior(gen_server).

%% API exports
-export([
         start_link/1,
         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 13: `src/at_server/at_extapi.erl`

```
1   %%%--------------------------------------------------------------------
```

```
%%% @doc
%%% Implementation of the atomic transaction server
%%% @end
%%%-------------------------------------------------------------------
%%% Student name: Rasmus Borgsmidt
%%% Student KU-id: qzp823
%%%-------------------------------------------------------------------

-module(at_extapi).

-export([abort/2, tryUpdate/2, ensureUpdate/2, choiceUpdate/3]).

%%%-------------------------------------------------------------------
%%% Extended API
%%%-------------------------------------------------------------------

%%--------------------------------------------------------------------
%% @doc
%% Aborts the specified transaction
%%
%% @spec abort(AT, Ref) -> aborted
%% where
%%   AT = pid()
%%   Ref = reference()
%% @end
%%--------------------------------------------------------------------
abort(AT, Ref) ->
    at_server:query_t(AT, Ref, fun(_) -> throw(abort) end).

%%--------------------------------------------------------------------
%% @doc
%% Tries to update the state on the specified server
%%
%% Returns: ok       if the state was updated successfully
%%          error    if the supplied function causes an error
%%          aborted  if another transaction is committed during the update
%%
%% @spec tryUpdate(AT, Fun) -> ok / error / aborted
%% where
%%   AT = pid()
%%   Fun = function(State)
%% @end
%%--------------------------------------------------------------------
tryUpdate(AT, Fun) ->
    {ok, TRef} = at_server:begin_t(AT),
    case at_server:query_t(AT, TRef, fun(S) -> S end) of
        aborted -> aborted;
        {ok, State} ->
            try Fun(State) of
                NewState ->
                    ok = at_server:update_t(AT, TRef, fun(_) -> NewState end),
                    at_server:commit_t(AT, TRef)
            catch
                _ : _ -> error
            end
    end.

%%--------------------------------------------------------------------
%% @doc
%% Tries to update the state on the specified server. This function will
%% keep trying until it succeeds, or the supplied function causes an error
%%
%% Returns: ok       if the state was updated successfully
%%          error    if the supplied function causes an error
%%
%% @spec ensureUpdate(AT, Fun) -> ok / error
%% where
%%   AT = pid()
%%   Fun = function(State)
%% @end
%%--------------------------------------------------------------------
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 14: `src/at_server/at_server_tests.erl`

```
-module(at_server_tests).
-include_lib("eunit/include/eunit.hrl").

-define(STATE, [1,2,3,4,5]).
-define(REV_STATE, [5,4,3,2,1]).

%%%-------------------------------------------------------------------
%%% Tests
%%%-------------------------------------------------------------------

doquery_success_test() ->
    AT = start(),
    {ok, ?STATE} = at_server:doquery(AT, fun id/1),
    stop(AT, ?STATE),
    [].

doquery_failure_test() ->
    AT = start(),
    error = at_server:doquery(AT, fun fail/1),
    stop(AT, ?STATE),
    [].

begin_t_test() ->
    AT = start(),
    {ok, _} = at_server:begin_t(AT),
    stop(AT, ?STATE),
    [].

query_t_success_test() ->
    AT = start(),
    {ok, TP} = at_server:begin_t(AT),
    {ok, ?STATE} = at_server:query_t(AT, TP, fun id/1),
    {ok, ?STATE} = at_server:query_t(AT, TP, fun id/1),
    stop(AT, ?STATE),
    [].

query_t_failure_test() ->
    AT = start(),
    {ok, TP} = at_server:begin_t(AT),
    aborted = at_server:query_t(AT, TP, fun fail/1),
    aborted = at_server:query_t(AT, TP, fun id/1),
    stop(AT, ?STATE),
    [].

update_t_success_test() ->
    AT = start(),
    {ok, TP1} = at_server:begin_t(AT),
    {ok, TP2} = at_server:begin_t(AT),
    {ok, ?STATE} = at_server:query_t(AT, TP1, fun id/1),
    ok = at_server:update_t(AT, TP1, fun reverse/1),
    {ok, ?REV_STATE} = at_server:query_t(AT, TP1, fun id/1),
    {ok, ?STATE} = at_server:doquery(AT, fun id/1),
    {ok, ?STATE} = at_server:query_t(AT, TP2, fun id/1),
    stop(AT, ?STATE),
    [].

update_t_failure_test() ->
    AT = start(),
    {ok, TP} = at_server:begin_t(AT),
    {ok, ?STATE} = at_server:query_t(AT, TP, fun id/1),
    ok = at_server:update_t(AT, TP, fun fail/1),
    aborted = at_server:query_t(AT, TP, fun id/1),
    {ok, ?STATE} = at_server:doquery(AT, fun id/1),
    stop(AT, ?STATE),
    [].

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 15: `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 16: 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
```

```
    at_server_tests: query_t_success_test...ok
    at_server_tests: doquery_failure_test...ok
    at_server_tests: doquery_success_test...ok
    [done in 0.087 s]
  [done in 0.087 s]
=======================================================
  All 10 tests passed.
ok
```

Listing 17: Session output: `src/at_server/at_extapi_tests.erl`

```
> eunit:test(at_extapi, [verbose]).
======================== EUnit ========================
module 'at_extapi'
  module 'at_extapi_tests'
    at_extapi_tests: choice_update_test...[0.001 s] ok
    at_extapi_tests: ensure_update_ok_test...ok
    at_extapi_tests: ensure_update_retry_test...[0.202 s] ok
    at_extapi_tests: ensure_update_error_test...ok
    at_extapi_tests: try_update_error_test...ok
    at_extapi_tests: try_update_aborted_test...[0.101 s] ok
    at_extapi_tests: try_update_ok_test...[0.001 s] ok
    at_extapi_tests: abort_test...ok
    [done in 0.328 s]
  [done in 0.328 s]
=======================================================
  All 8 tests passed.
ok
```

Listing 18: Session output: `commit_t_competing_test()` (debug enabled)

```
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31695> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31699> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31704> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31714> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31719> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31723> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31728> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31732> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31736> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31740> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31744> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31748> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31758> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31762> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31768> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31772> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31776> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31780> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31784> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31788> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31790> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31792> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31794> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31796> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31800> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31806> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31810> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31814> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31818> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31822> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31826> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31830> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31832> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31834> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31836> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31840> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31844> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31848> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31852> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31856> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31860> aborted
```

```
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31864> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31868> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31872> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31874> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31876> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31878> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31882> aborted
at_server_tests.erl:154:<0.3764.0>: ----> Transaction #Ref<0.0.0.31680> committed
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31682> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31684> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31686> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31688> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31691> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31693> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31697> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31701> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31706> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31708> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31710> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31712> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31717> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31721> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31725> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31730> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31734> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31738> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31742> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31746> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31750> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31752> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31754> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31756> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31760> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31764> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31766> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31770> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31774> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31778> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31782> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31786> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31798> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31802> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31804> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31808> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31812> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31816> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31820> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31824> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31828> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31838> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31842> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31846> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31850> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31854> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31858> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31862> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31866> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31870> aborted
at_server_tests.erl:151:<0.3764.0>: Transaction #Ref<0.0.0.31880> aborted
```

D SUBMITTED FILE TREE

Listing 19: File tree under `src/`

```
src/
|-- at_server
|   |-- at_extapi.erl
|   |-- at_extapi_tests.erl
|   |-- at_server.erl
|   |-- at_server_tests.erl
|   +-- at_trans.erl
+-- salsa
    |-- Gpx.hs
    |-- SalsaAst.hs
    |-- SalsaInterp.hs
    |-- SalsaParser.hs
    |-- SalsaParserTest.hs
    |-- SimpleParse.hs
    |-- multi.salsa
    +-- simple.salsa
```