ANATOMY OF AN X-GRIN BACK END

Andor Penzes

URBAN BOQUIST'S THESIS

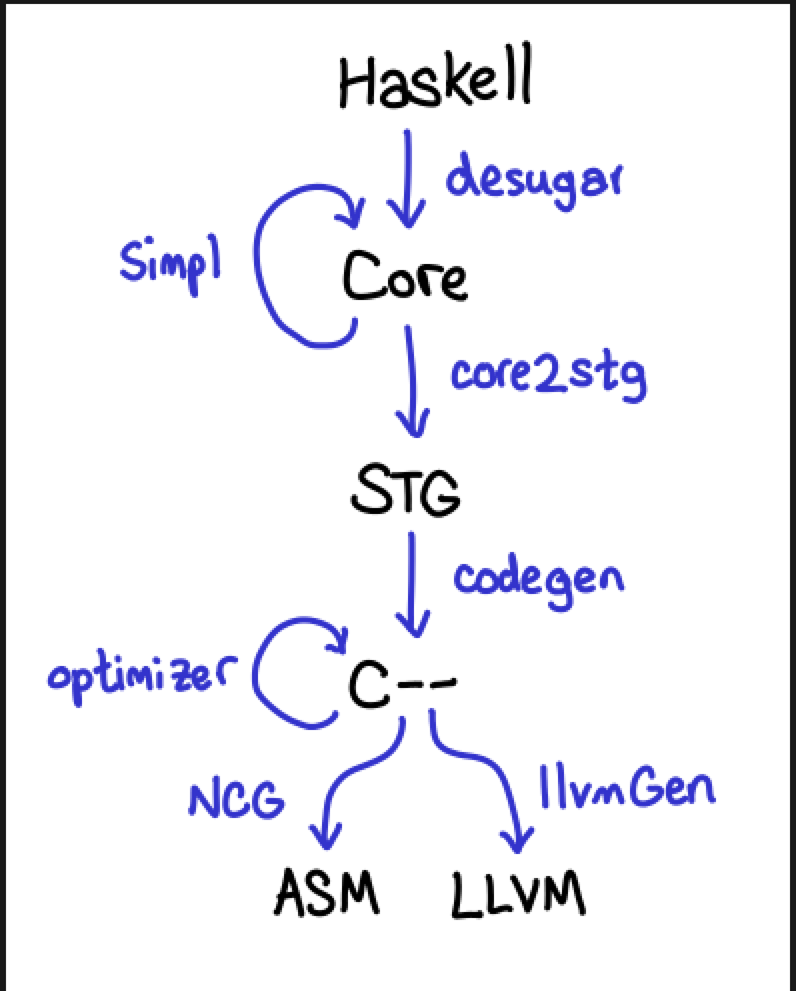
- Simple C like language: GRIN
- Lazy computation via explicit HEAP objects
- Program transformations based on whole program analysis
- Lambda calculus like language: Lambda
- Translation from Lambda to GRIN

GRIN PROJECT

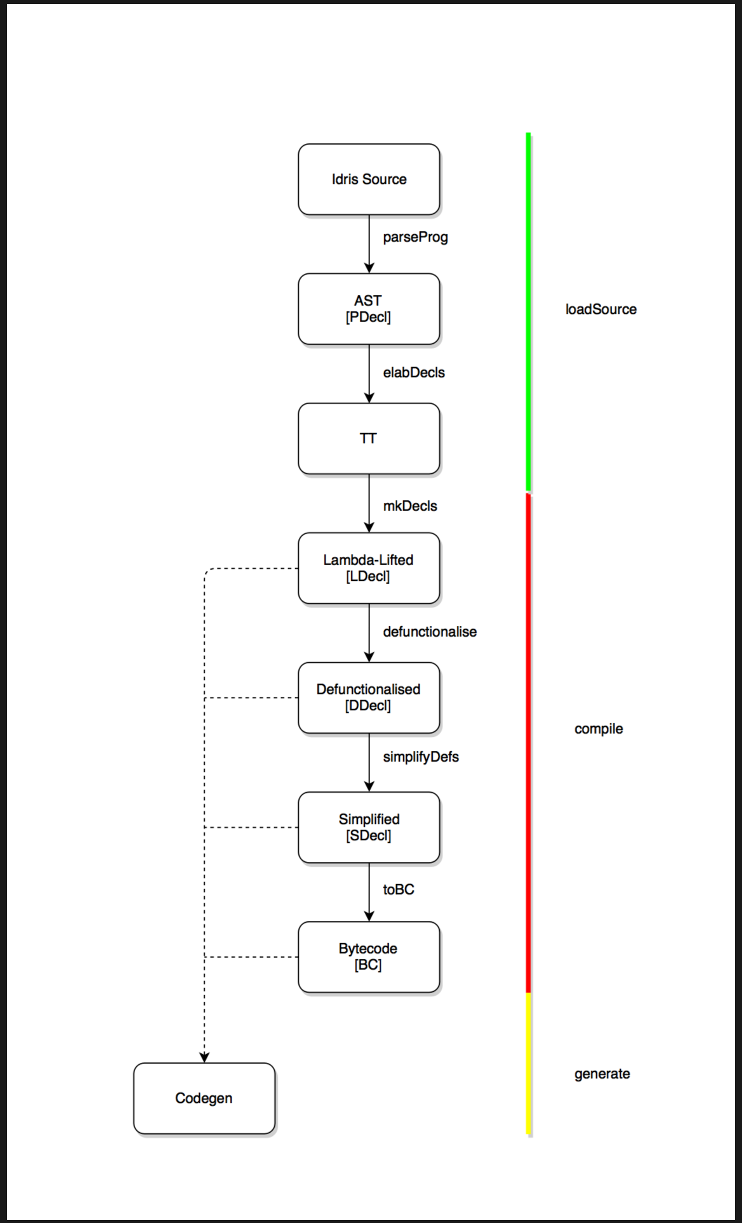
- Our goal is to write a unified compiler back end for lazy and non-lazy functional programming languages.
- We actively develop two GRIN back ends, one for GHC and the Idris. Meanwhile we implement the GRIN-compiler. After finishing the Idris we start to work on the Agda GRIN back end.

COMPILER PIPELINES

GHC



IDRIS



Idris Source
parseProg
AST
[PDecl]
elabDecls
TT
mkDecls
Lambda-Lifted
[LDecl]
defunctionalise
Defunctionalised
[DDecl]
simplifyDefs
Simplified
[SDecl]
toBC
Bytecode
[BC]
Codegen
loadSource
compile
generate

SYNTAX

LAMBDA

```
data Exp
  = Program     [External] [Def]
  | Def         Name [Name] Exp

  | App         Name [Atom]
  | Let         [(Name, Exp)] Exp -- lazy let
  | LetRec      [(Name, Exp)] Exp -- recursive lazy let
  | LetS        [(Name, Exp)] Exp -- strict let
  | Con         Name [Atom]

  | Case        Atom [Alt]
  | Alt         Pat Exp

  | Var         Bool Name -- is pointer
  | Lit         Lit

  | Closure     [Name] [Name] Exp
```

GRIN

```haskell
data Exp
  = Program     [External] [Def]
  | Def         Name [Name] Exp
  -- Exp
  | EBind       SimpleExp LPat Exp
  | ECase       Val [Alt]
  -- Simple Exp
  | SApp        Name [SimpleVal]
  | SReturn     Val
  | SStore      Val
  | SFetch      Name
  | SUpdate     Name Val
  | SBlock      Exp
  -- Alt
  | Alt CPat Exp
```

GRIN PROJECT

- GRIN compiler
- GHC-GRIN backend
- Idris-GRIN backend

GHC-GRIN BACKEND

- Compiles STG to Lambda
- Generates GRIN from Lambda

GHC-GRIN BACKEND

- Compiles STG to Lambda
- Generates GRIN from Lambda

Challenge:

- The size of the GHC hello world is large
- The whole program analysis described in PhD thesis is not powerful enough

GHC-GRIN BACKEND

- Compiles STG to Lambda
- Generates GRIN from Lambda

Challenge:

- The size of the GHC hello world is large
- The whole program analysis described in PhD thesis is not powerful enough

Ongoing work:

- Abstract interpretation of STG using Suffle (https://souffle-lang.github.io/)
- Interprocedural dead-code elimination on STG level
- Improved GRIN code generation based on Abstract Interpretation

IDRIS-GRIN BACKEND

- End-to-end testing for the GRIN compiler

IDRIS-GRIN BACKEND

- End-to-end testing for the GRIN compiler
- More fun learning Idris than writing test-case programs in GRIN

IDRIS-GRIN BACKEND

- End-to-end testing for the GRIN compiler
- More fun learning Idris than writing test-case programs in GRIN
- The implementation is based on examples of the TDDI book

IDRIS-GRIN BACKEND

- End-to-end testing for the GRIN compiler
- More fun learning Idris than writing test-case programs in GRIN
- The implementation is based on examples of the TDDI book
- Consumes the simplest Idris IR

IDRIS-GRIN BACKEND

- End-to-end testing for the GRIN compiler
- More fun learning Idris than writing test-case programs in GRIN
- The implementation is based on examples of the TDDI book
- Consumes the simplest Idris IR
- Introduce challenges like FFI, runtime, and garbage collection

WHY DEVELOP THREE THINGS AT THE SAME TIME?

- Pragmatic
- An Incremental Approach to Compiler Construction [1]
- Restrictions, requirements, and test cases are created based on real compilers

[1] http://scheme2006.cs.uchicago.edu/11-ghuloum.pdf

IDRIS-GRIN BACKEND

- Standalone executable which is invoked by the Idris compiler via the --codegen option
- GRIN code generation from Simplified SDecl
- Glue code between Idris-GRIN and C
- Primitive operations implemented in C
- Simple Runtime in C which needs a lot of improvements

```
exp fname = \case
  SOp f lvars                              -> primFn f (map (Var . lvar fname) lvars)
  scon@(SCon maybeLVar int name lvars)  -> SReturn $ val fname scon
  sconst@(SConst cnst)                     -> SReturn $ val fname sconst

  Idris.SApp bool nm lvars -> Grin.SApp (name nm) (map (Var . lvar fname) lvars)

  SLet loc0@(Idris.Loc i) v sc ->
    EBind (SBlock (exp fname v)) ((localName fname loc0) ++ "_val") $
    EBind (SStore (localName fname loc0) ++ "_val") (localName fname loc0) $
    (exp fname sc)

  Idris.SUpdate loc0 exp0 ->
    EBind (SBlock (exp fname exp0)) (localName fname loc0 ++ "_val") $
    EBind (Grin.SUpdate (loc fname loc0)) (localName fname loc0) ++ "_val") Unit $
    SReturn localName fname loc0) ++ "_val")

  SCase caseType lvar0 salts ->
    EBind (SFetch (lvar fname lvar0)) (varName fname lvar0) ++ "_val")) $
    ECase (varName fname lvar0) ++ "_val") (alts fname salts)

  SChkCase lvar0 salts ->
    EBind (SFetch $ varName fname lvar0)
                    (varName fname lvar0 ++ "_val") $
    ECase (vanName fname lvar0) ++ "_val") (alts fname salts)

  SV lvar0@(Idris.Loc i)  -> SFetch (localName fname lvar0)
  SV lvar0@(Idris.Glob n) -> Grin.SApp (varName fname lvar0) []
```

GRIN code generation from Simplified SDecl

```
primFn :: Idris.PrimFn -> [SimpleVal] -> Exp
primFn f ps = case f of
  LPlus   (Idris.ATInt intTy) -> Grin.SApp "idris_int_add" ps
  LPlus   Idris.ATFloat       -> Grin.SApp "idris_float_add" ps
  LMinus  (Idris.ATInt intTy) -> Grin.SApp "idris_int_sub" ps
  LMinus  Idris.ATFloat       -> Grin.SApp "idris_float_sub" ps
  LTimes  (Idris.ATInt intTy) -> Grin.SApp "idris_int_mul" ps
  LTimes  Idris.ATFloat       -> Grin.SApp "idris_float_mul" ps
  LSDiv   (Idris.ATInt intTy) -> Grin.SApp "idris_int_div" ps
  LSDiv   Idris.ATFloat       -> Grin.SApp "idris_float_div" ps
  ...
```

Glue code between Idris-GRIN and C (1)

```
idris_int_eq idris_int_eq0 idris_int_eq1 =
  (CGrInt idris_int_eq0_1) <- fetch idris_int_eq0
  (CGrInt idris_int_eq1_1) <- fetch idris_int_eq1
  idris_int_eq2 <- _prim_int_eq idris_int_eq0_1 idris_int_eq1_1
  case idris_int_eq2 of
    #False  -> pure (CGrInt 0)
    #True   -> pure (CGrInt 1)

idris_float_eq idris_float_eq0 idris_float_eq1 =
  (CGrFloat idris_float_eq0_1) <- fetch idris_float_eq0
  (CGrFloat idris_float_eq1_1) <- fetch idris_float_eq1
  idris_float_eq2 <- _prim_float_eq idris_float_eq0_1 idris_float_eq1_1
  case idris_float_eq2 of
    #False  -> pure (CGrInt 0)
    #True   -> pure (CGrInt 1)

idris_write_str idris_write_str1 idris_write_str2 =
  (CGrString idris_write_str2_0) <- fetch idris_write_str2
  _prim_string_print idris_write_str2_0
  pure (CUnit)

idris_time =
  idris_time1 <- _prim_time
  pure (CGrInt idris_time1)
```

Glue code between Idris-GRIN and C (2)

```
primop pure
  _prim_int_eq    :: T_Int64 -> T_Int64 -> T_Bool
  _prim_int_add   :: T_Int64 -> T_Int64 -> T_Int64
  _prim_int_sub   :: T_Int64 -> T_Int64 -> T_Int64
  _prim_int_mul   :: T_Int64 -> T_Int64 -> T_Int64
  _prim_int_div   :: T_Int64 -> T_Int64 -> T_Int64

  _prim_float_eq  :: T_Float -> T_Float -> T_Bool
  _prim_float_add :: T_Float -> T_Float -> T_Float
  _prim_float_sub :: T_Float -> T_Float -> T_Float
  _prim_float_mul :: T_Float -> T_Float -> T_Float
  _prim_float_div :: T_Float -> T_Float -> T_Float

primop effectful
  _prim_string_print  :: T_String -> T_Unit
  _prim_usleep        :: T_Int64 -> T_Unit
  _prim_time          :: T_Int64
```

Primitive operations implemented in C

```c
void _prim_string_print(struct string* p1){
    for(int i = 0; i < p1->length; i++) {
        putchar(p1->data[i]);
    }
}

void _prim_usleep(int64_t p1) {
    usleep(p1); // p1 microseconds
}

int64_t _prim_time() {
    time_t t = time(NULL);
    return (int64_t)t;
}
```

Simple Runtime in C which needs a lot of improvements

```c
extern int64_t _heap_ptr_;
int64_t grinMain();

void __runtime_error(int64_t code){
  exit(code);
}

int main() {
  int64_t* heap = malloc(100*1024*1024);
  _heap_ptr_ = (int64_t)heap;
  grinMain();
  free(heap);
  return 0;
}
```

HELLO WORLD IN IDRIS-GRIN

- Idris
- Compiled GRIN
- Optimised GRIN

Hello World - Idris

```
module Main

main : IO ()
main = putStrLn "Hello World!"
```

```
grinMain =
  r <- idr_{runMain_0}
  pure ()

idr_{runMain_0} =
  v.3 <- pure (CErased)
  idr_{runMain_0}0_val_5 <- pure v.3
  idr_{runMain_0}0 <- store idr_{runMain_0}0_val_5
  idr_{runMain_0}0_val <- idr_Main.main idr_{runMain_0}0
  idr_{runMain_0}0_6 <- store idr_{runMain_0}0_val
  idr_{EVAL_0} idr_{runMain_0}0_6

idr_{EVAL_0} idr_{EVAL_0}0 =
  idr_{EVAL_0}0_val <- fetch idr_{EVAL_0}0
  fetch idr_{EVAL_0}0

idr_Main.main idr_Main.main0 =
  v.1 <- pure (CGrString #"Hello World!\n")
  idr_Main.main1_val_3 <- pure v.1
  idr_Main.main1 <- store idr_Main.main1_val_3
  idr_Main.main1_val <- idris_write_str idr_Main.main0 idr_Main.main1
  idr_Main.main1_4 <- store idr_Main.main1_val
  pure (Cidr_MkUnit)

idris_write_str idris_write_str1 idris_write_str2 =
  (CGrString idris_write_str2_0) <- fetch idris_write_str2
  _prim_string_print $ idris_write_str2_0
  pure (CUnit)
```

Hello World - GRIN optimised

```
grinMain =
  idr_Main.main1.33.0.arity.1.0 <- pure #"Hello World!\n"
  _prim_string_print idr_Main.main1.33.0.arity.1.0
```

PRELIMINARY RESULTS

```
 60416 01_Average.idr.bin
 13704 01_Average.idr.grin.bin

136384 01_DataTypes.idr.bin
 28960 01_DataTypes.idr.grin.bin

 55728 01_ExactLength.idr.bin
 11320 01_ExactLength.idr.grin.bin

 85640 01_Exercises.idr.bin
 15664 01_Exercises.idr.grin.bin

 27328 01_HelloWorld.idr.bin
  9776 01_HelloWorld.idr.grin.bin
```

QUESTIONS?

- https://github.com/grin-compiler/grin
- https://github.com/grin-compiler/idris-grin
- https://github.com/grin-compiler/ghc-grin
- https://www.patreon.com/csaba_hruska