ANATOMY OF AN X-GRIN BACK END

Andor Penzes

URBAN BOQUIST'S THESIS

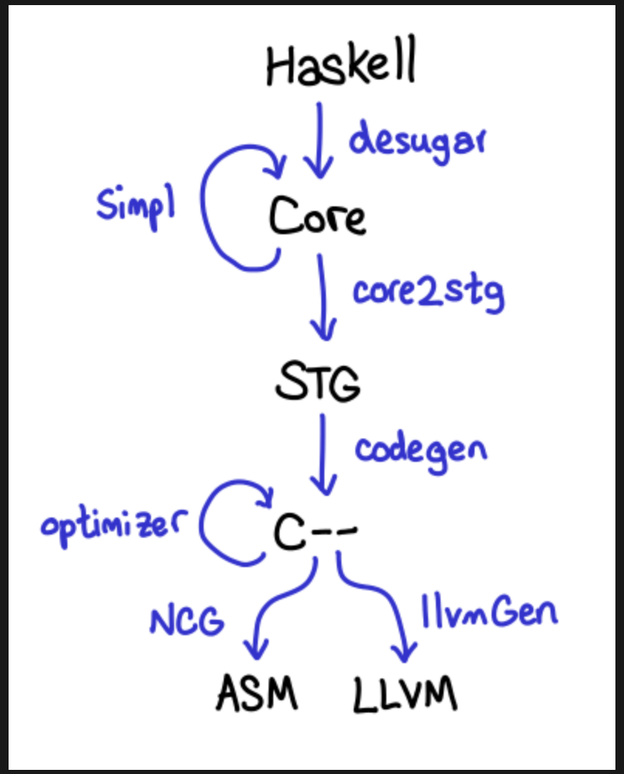
- Simple C like language: GRIN
- Lazy computation via explicit HEAP objects
- Program transformations based on whole program analysis
- Lambda calculus like language: Lambda
- Translation from Lambda to GRIN

GRIN PROJECT

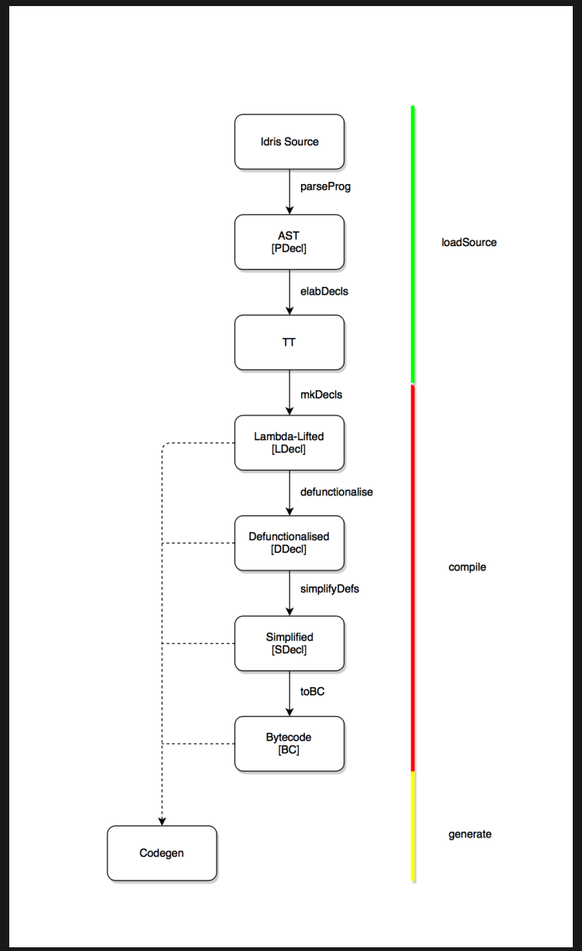
- Our goal is to write a unified compiler back end for lazy and non-lazy functional programming languages.
- We actively develop two GRIN based back ends, one for GHC and one for Idris meanwhile implementing the GRIN-compiler.

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
```

Haskell Source

```
upto n m | n > m     = []
         | otherwise = n : (upto (n+1) m)
```

Lambda Source

```
Main.$wupto ww.s16190.0 ww1.s16191.0 =
  letS lwild.s16192.0 = ># $ ww.s16190.0 ww1.s16191.0
  case lwild.s16192.0 of
    _ ->
        let sat.s16195.0 = \[ww.s16190.0 ww1.s16191.0]  ->
              letS sat.s16194.0 = +# $ ww.s16190.0 1
              Main.$wupto sat.s16194.0 ww1.s16191.0
        let sat.s16193.0 = [GHC.Types.I# ww.s16190.0]
        [GHC.Types.: sat.s16193.0 sat.s16195.0]
    1 ->
        [GHC.Types.[]]
```

GRIN

```
data Exp
  = Program     [External] [Def]
  | Def         Name [Name] Exp
  -- Exp
  | Bind        Exp LPat Exp
  | Case        Val [Alt]
  | Alt         CPat Exp
  -- Simple Exp
  | App         Name [SimpleVal]
  | Pure        Val
  | Store       Val
  | Fetch       Name
  | Update      Name Val
  | Block       Exp
```

GRIN SOURCE

```
upto m n = (CInt m') <- eval m
           (CInt n') <- eval n
           b <- _prim_int_gt m' n'
           case b of
             #True -> pure (CNil)
             #False ->
               m1' <- _prim_int_add m' 1
               m1 <- store (CInt m1')
               p <- store (Fupto m1 n)
               pure (CCons m p)
```

GRIN PROJECT

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

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 test cases are 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 test cases are 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 test cases are based on examples of the TDDI book
- Consumes the simplest Idris IR
- Introduce challenges like FFI, runtime, and garbage collection

IDRIS-GRIN BACKEND

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

GRIN code generation from SDecl

```
I.SLet loc0@(I.Loc{}) v sc ->
  G.Bind (G.Block (sexp fname v)) (varV loc0) $
  G.Bind (G.Store (varV loc0))    (var loc0)  $
  (sexp fname sc)

I.App bool nm lvars -> G.App (name nm) (map var lvars)

I.SUpdate loc0 sexp0 ->
  G.Bind (G.Block (sexp fname sexp0)) (varV loc0) $
  G.Bind (G.Update (variableName $ var loc0) (varV loc0)) G.Unit $
  G.Pure (varV loc0)

I.SCase caseType lvar0 salts ->
  G.Bind (G.Fetch $ variableName $ var lvar0) (varV lvar0) $
  G.Case (varV lvar0) (alts fname salts)

I.SOp f lvars -> primFn f (map var lvars)

sc@(I.SCon{})   -> G.Pure $ val fname sc
sc@(I.SConst{}) -> G.Pure $ val fname sc

I.SV lvar0@(I.Loc{})  -> G.Fetch $ variableName $ var lvar0
I.SV lvar0@(I.Glob{}) -> G.App $ variableName $ var lvar0) []
```

GRIN code generation from SDecl

```
primFn :: I.PrimFn -> [G.SimpleVal] -> G.Exp
primFn f ps = case f of
  I.LPlus   (I.ATInt _) -> G.SApp "idris_int_add" ps
  I.LPlus   I.ATFloat   -> G.SApp "idris_float_add" ps
  I.LMinus  (I.ATInt _) -> G.SApp "idris_int_sub" ps
  I.LMinus  I.ATFloat   -> G.SApp "idris_float_sub" ps
  I.LTimes  (I.ATInt _) -> G.SApp "idris_int_mul" ps
  I.LTimes  I.ATFloat   -> G.SApp "idris_float_mul" ps
  I.LSDiv   (I.ATInt _) -> G.SApp "idris_int_div" ps
  I.LSDiv   I.ATFloat   -> G.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_float_eq      :: T_Float -> T_Float -> T_Bool
  _prim_float_add     :: T_Float -> T_Float -> T_Float

ffi 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!"
```

```
idris HelloWorld.idr --codegen grin -o helloworld.bin
```

```
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 =
  x <- pure #"Hello World!\n"
  _prim_string_print x
```

PRELIMINARY RESULTS

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

107832 01_Interfaces.idr.bin
 25048 01_Interfaces.idr.grin.bin

150368 01_IOIntro.idr.bin
 25136 01_IOIntro.idr.grin.bin

243312 02_Game.idr.bin
 32136 02_Game.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

MEMORY MANAGEMENT

- LLVM supports GC: robinvd/lang-experiments/
- Counting Immutable Beans: arxiv.org/abs/1908.05647
- ASAP memory management