

GHC Reading Guide

- Exploring entrances and mental models to source code -

Takenobu T.

NOTE:

- This is not an official document by the ghc development team.
- Please refer to the official documents in detail.
- Don't forget "semantics". It's very important.
- This is written for ghc 9.0.

Contents

Introduction

1. Compiler

- Compilation pipeline
- Each pipeline stages
- Intermediate language syntax
- Call graph

2. Runtime system

3. Core libraries

Appendix

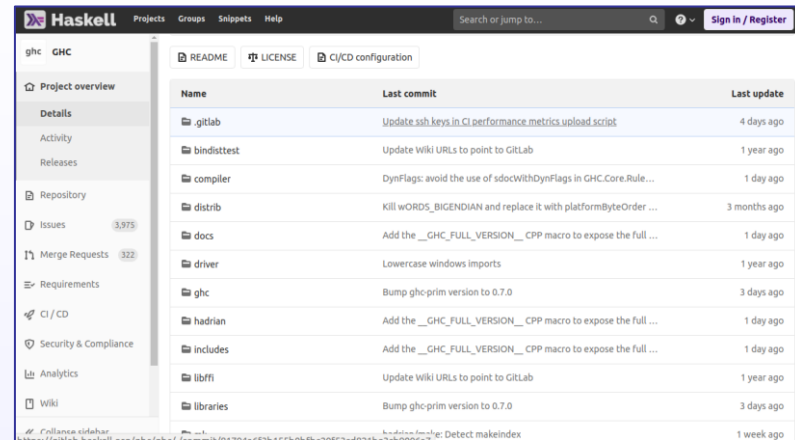
References

Introduction

Official resources are here

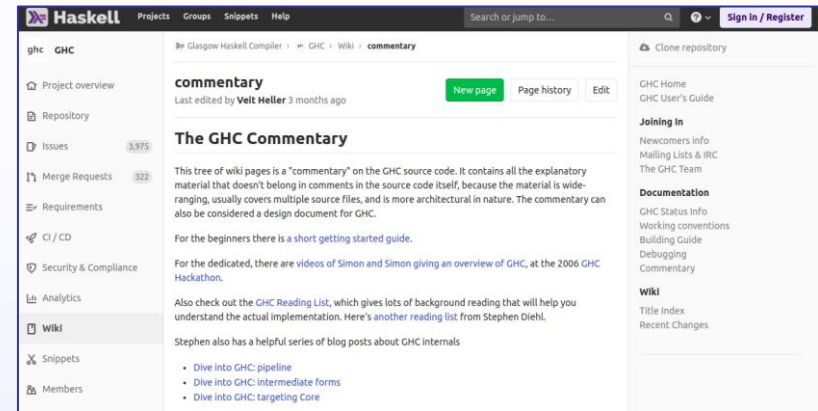
GHC source repository :

<https://gitlab.haskell.org/ghc/ghc>



The GHC Commentary (for developers) :

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary>



GHC Documentation (for users) :

* master HEAD

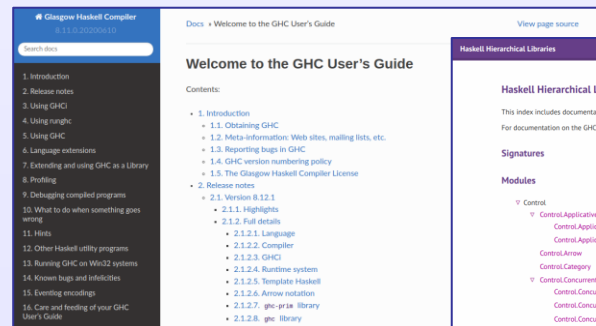
<https://ghc.gitlab.haskell.org/ghc/doc/>

* latest major release

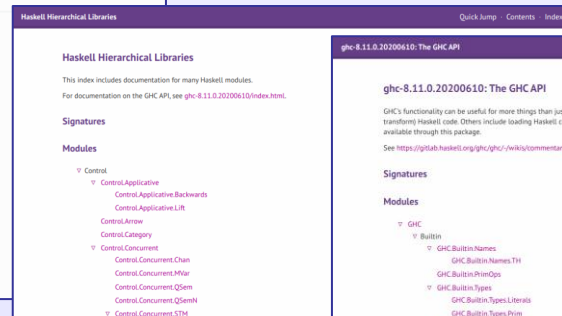
<https://downloads.haskell.org/~ghc/latest/docs/html/>

* version specified

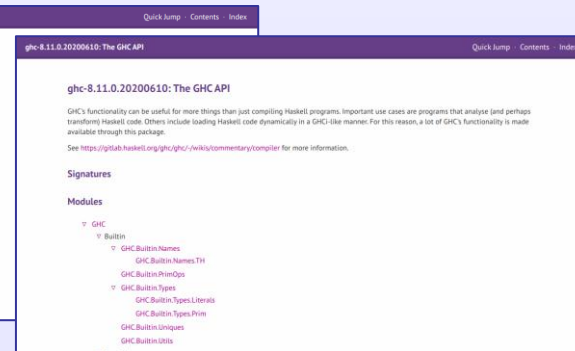
<https://downloads.haskell.org/~ghc/9.0.1/docs/html/>



The User's Guide

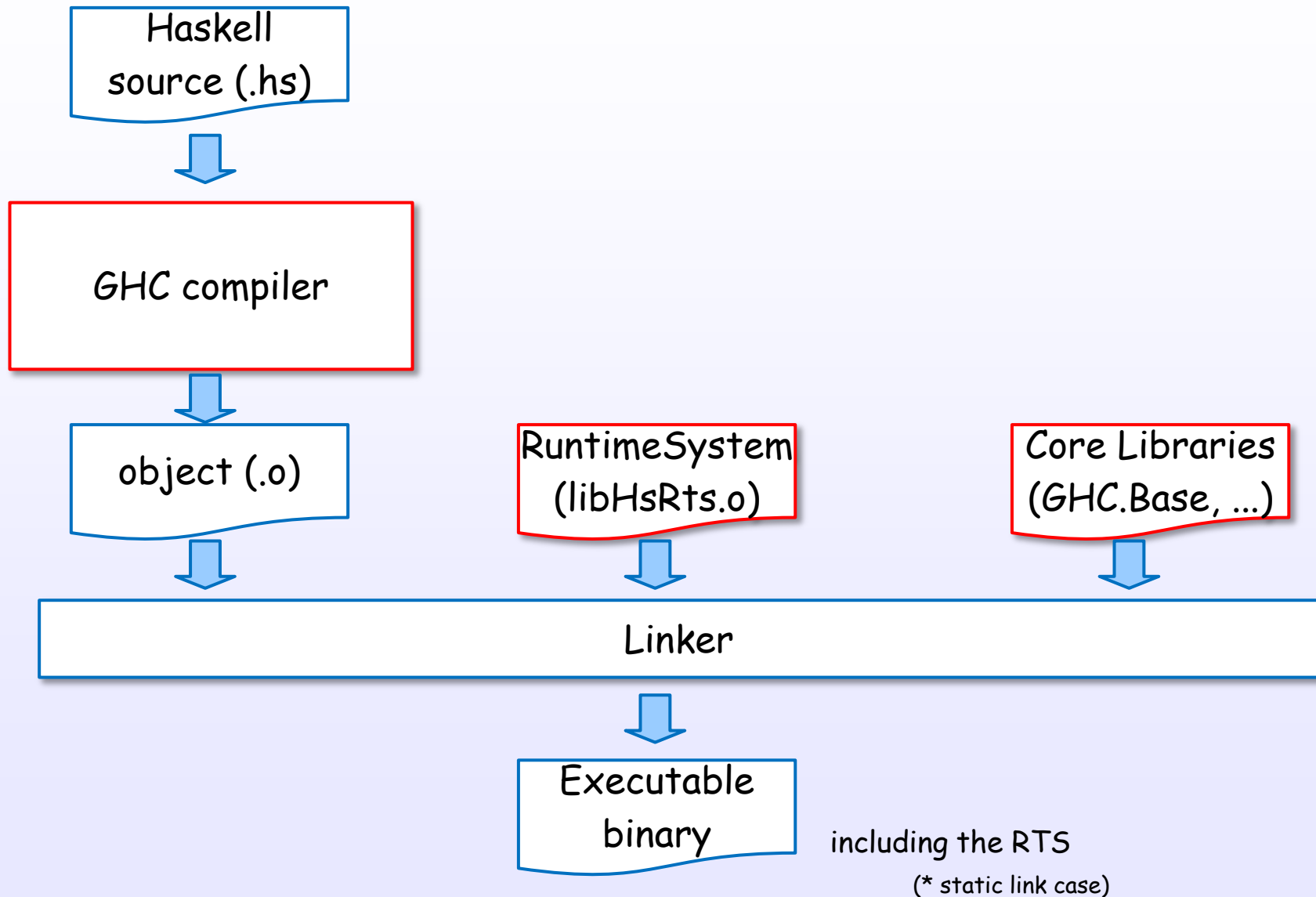


Core Libraries



GHC API

The GHC = Compiler + Runtime System (RTS) + Core Libraries



Each division is located in the GHC source tree

GHC source repository :

<https://gitlab.haskell.org/ghc/ghc>

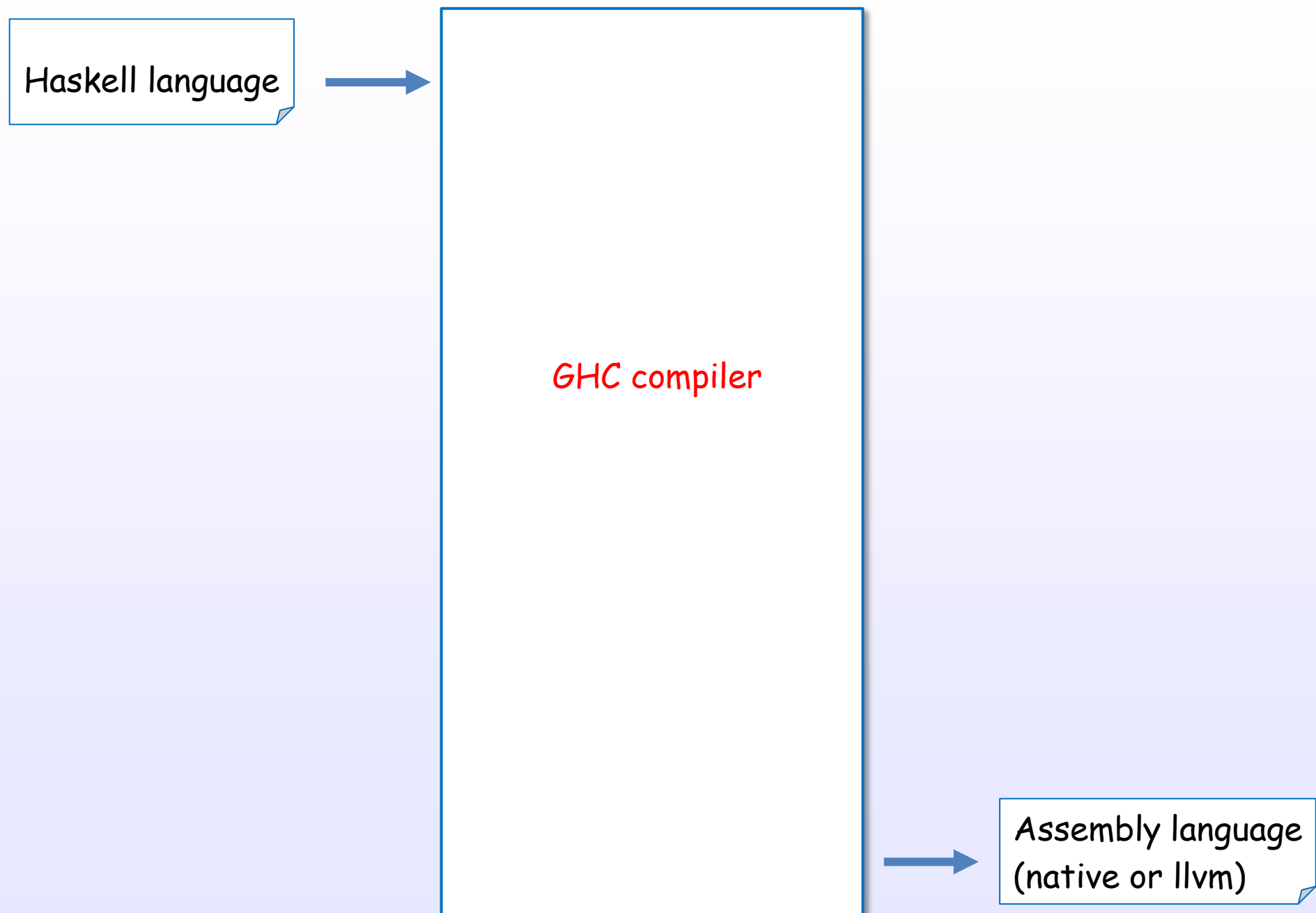
compiler/	... compiler sources
rts/	... runtime system sources
libraries/	... core library sources
ghc/	... compiler main
includes/	... include files
testsuite/	... test suites
nofib/	... performance tests
mk/	... build system
hadrian/	... hadrian build system
docs/	... documents
:	:

1. Compiler

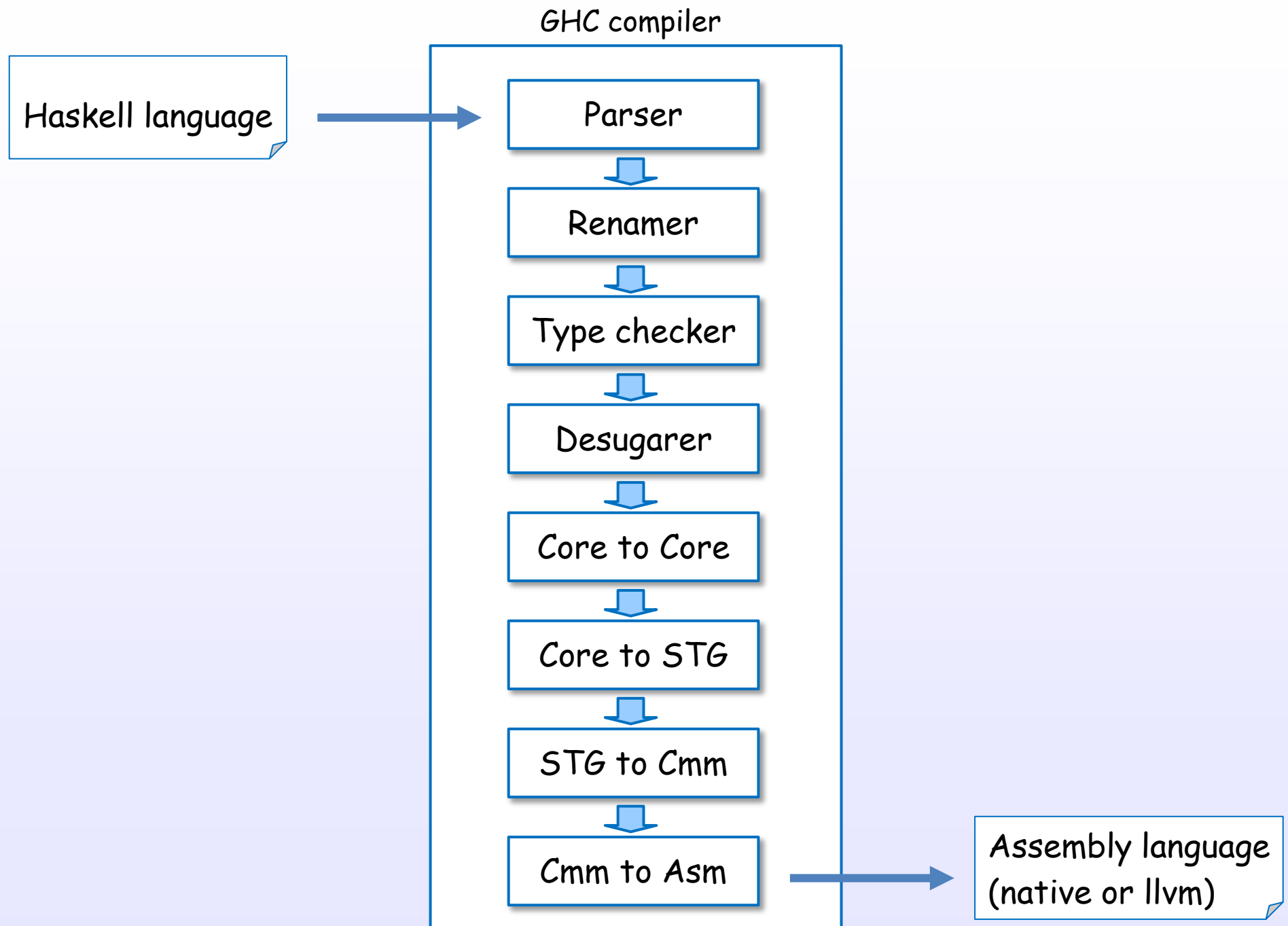
1. Compiler

Compilation pipeline

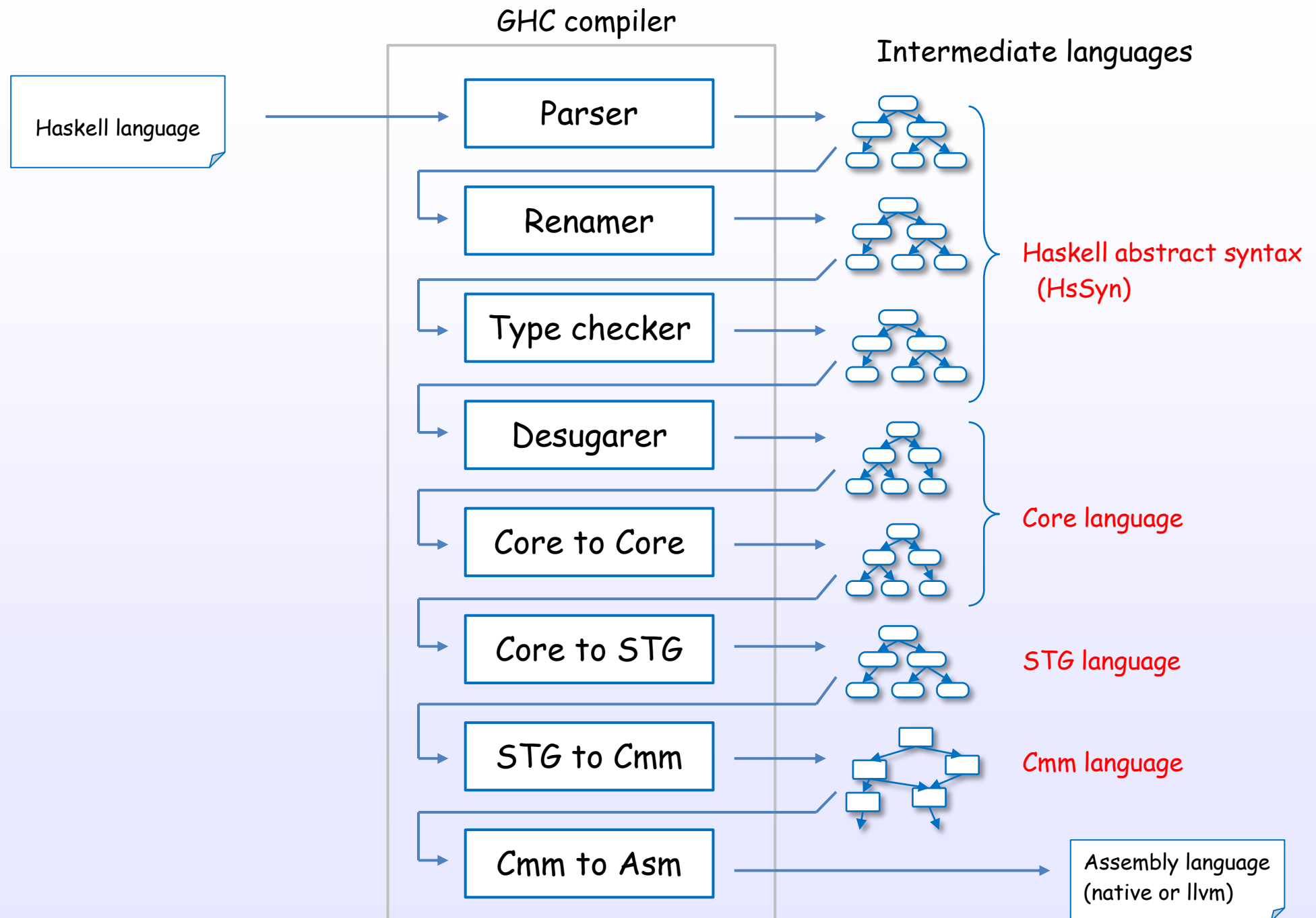
The GHC compiler



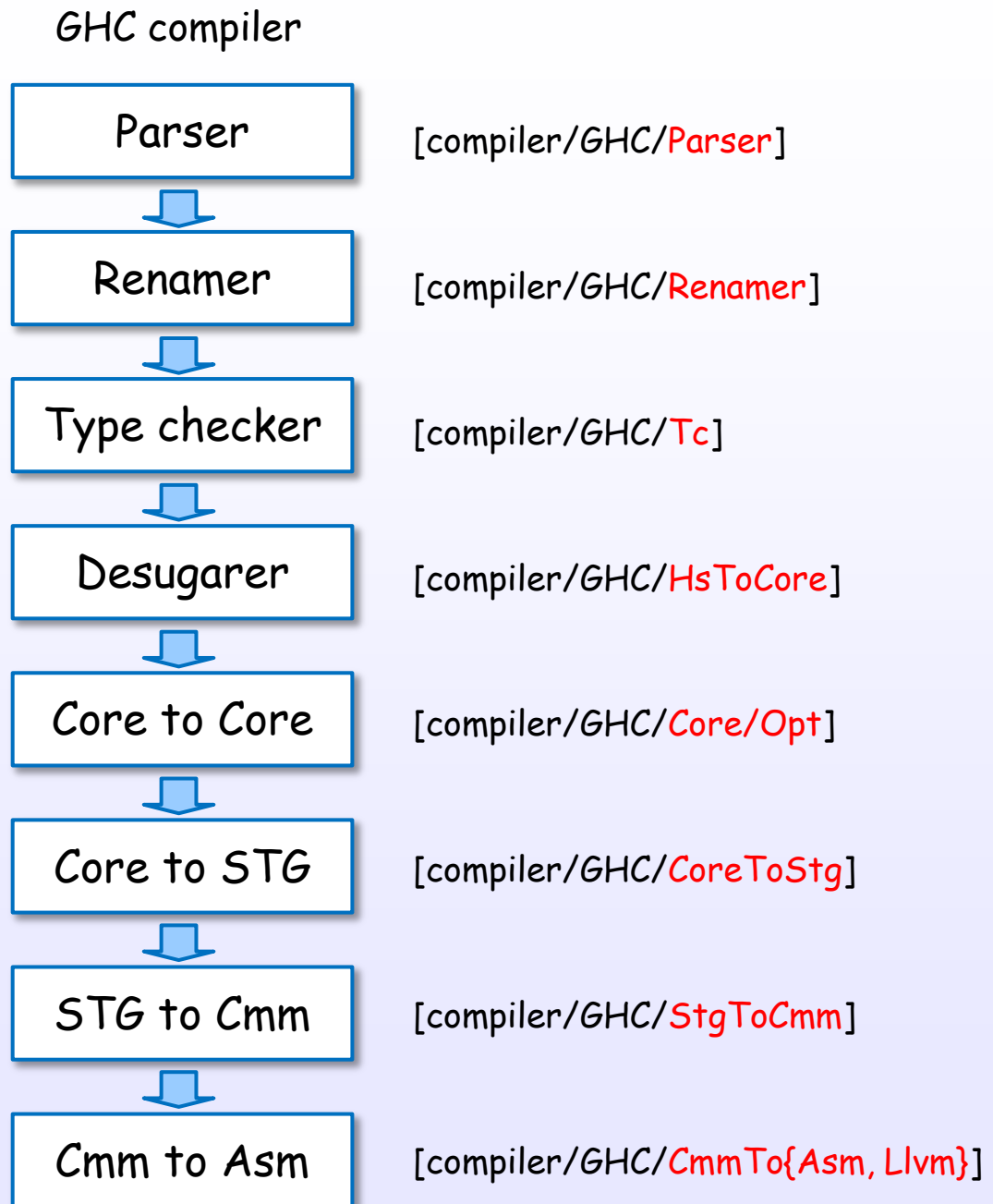
GHC's Haskell compilation pipeline



Pipeline stages with intermediate languages



Each stages is implemented in



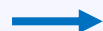
1. Compiler

Each pipeline stages

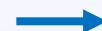
Parser

Haskell source

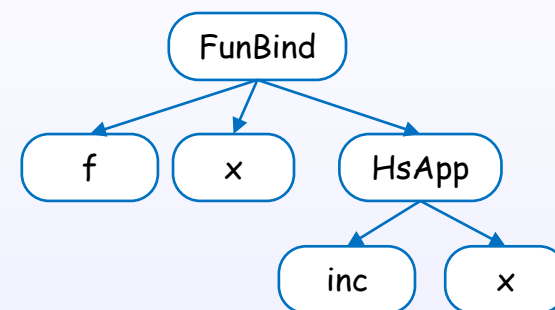
```
f x = inc x
```



Parser
(reader)



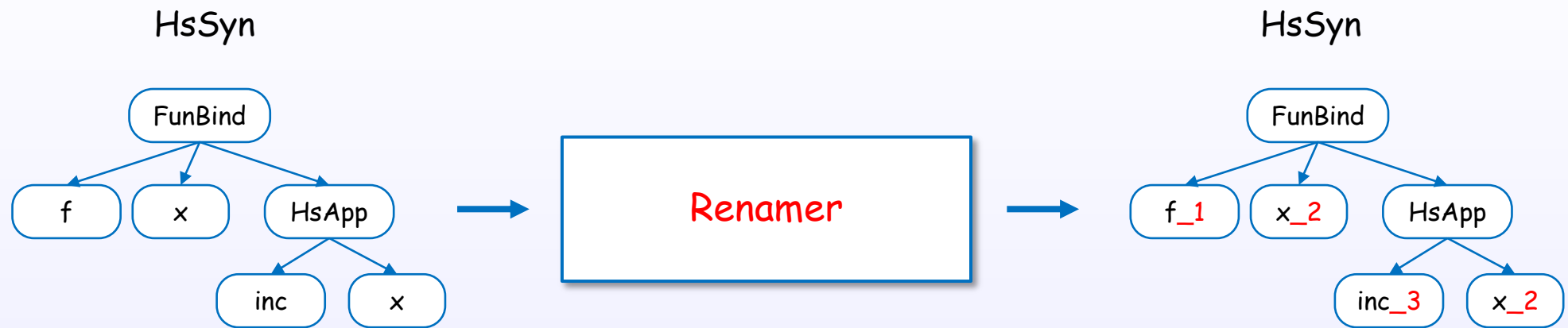
HsSyn
(Haskell Abstract Syntax)



Abstracted

- Parsing a Haskell source file
- Checking user syntax errors
- etc.

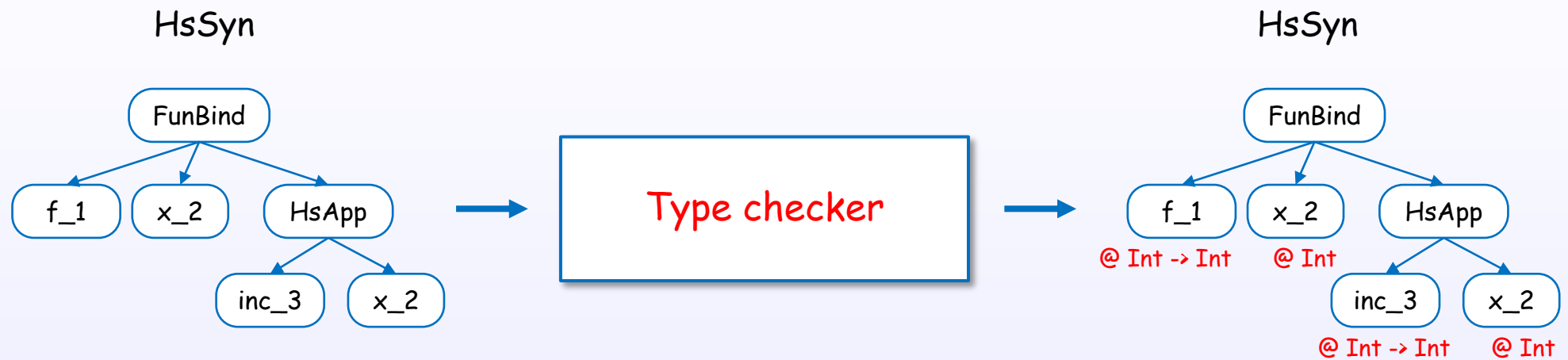
Renamer



Unique named

- Resolving all of the identifiers
- Rearranging infix expressions
- Checking user scope errors
- Building global environments
- etc.

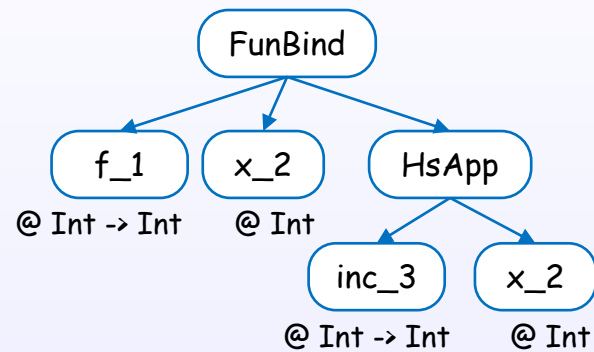
Type checker



- Resolving/Inferring types
- Decorating AST with type
- Checking user type errors
- etc.

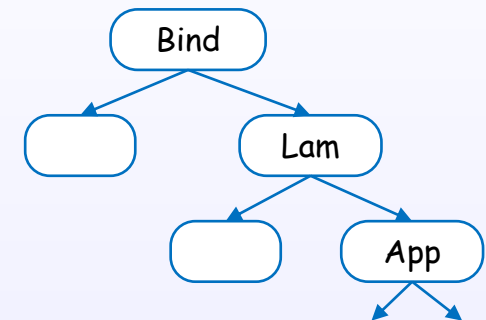
Desugarer

HsSyn



Desugarer
(HsToCore)

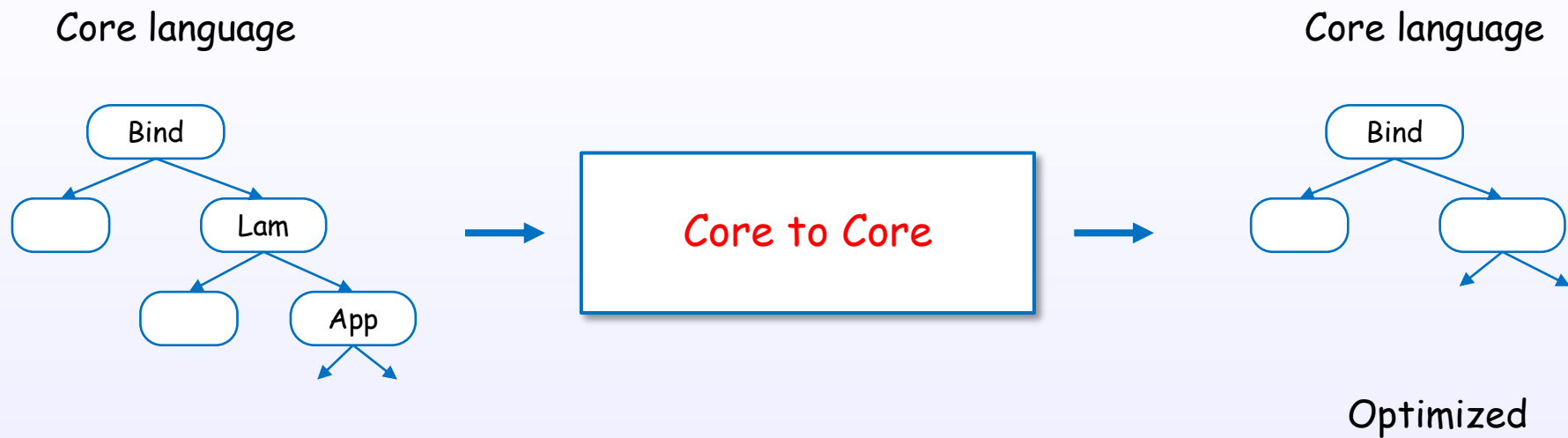
Core language



Squeezed into λ calculus

- Desugaring HsSyn to Core
- Checking pattern-match overlap
- etc.

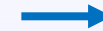
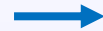
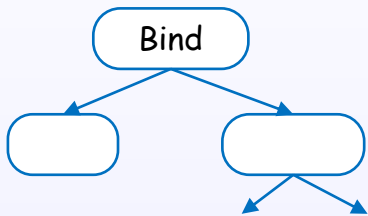
Core to Core



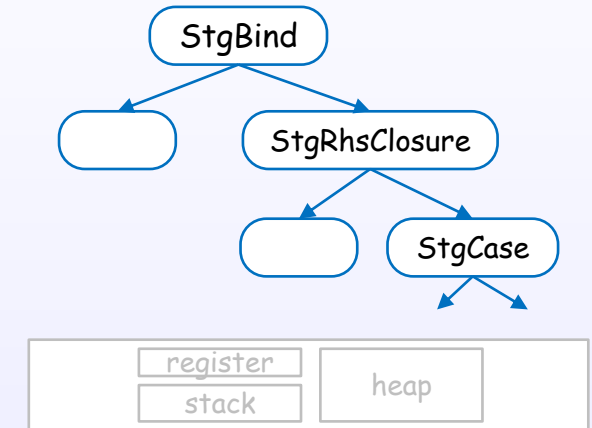
- Optimizing Core (simplifier, ...)
- Checking typechecker's result with Lint
- Tidying Core to Core
- etc.

Core to Stg

Core language



STG language

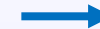
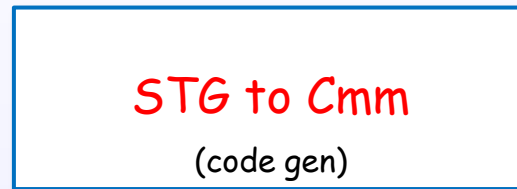
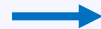
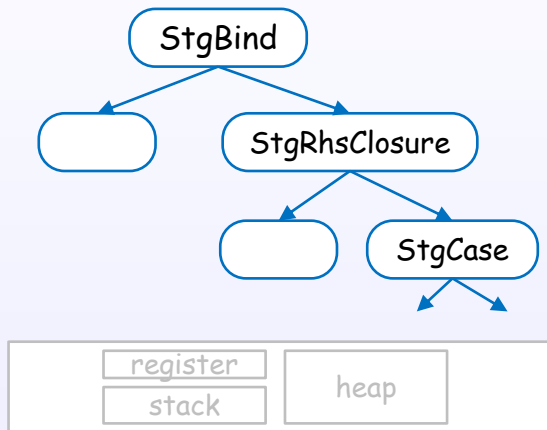


- Transforming Core to STG
- etc.

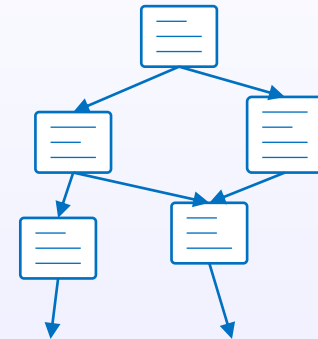
Operationally mapped

STG to Cmm

STG language



Cmm language

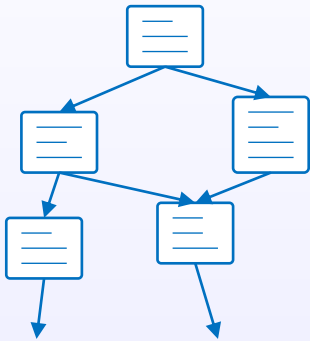


Instruction blocked

- Optimizing STG
- Transforming STG to Cmm
- etc.

Cmm to Assembly

Cmm language



Assembly/LLVM language

```
add  $0x2, %rbx
jmpq *0x0(%rbp)
:
```

Machine coded

- Optimizing Cmm
- Generating Asm
- etc.

1. Compiler

Intermediate language syntax

HsSyn (Haskell abstract syntax)

[compiler/GHC/Hs/Decls.hs]

```
data HsDecl p
  = TyCID ...           -- Type or Class Declaration
  | InstD ...           -- Instance declaration
  | DerivD ...          -- Deriving declaration
  | ValD ...            -- Value declaration
  | SigD ...            -- Signature declaration
  | KindSigD ...        -- Standalone kind signature
  | DefD ...            -- 'default' declaration
  | ForD ...            -- Foreign declaration
  | WarningD ...        -- Warning declaration
  | AnnD ...            -- Annotation declaration
  | RuleD ...           -- Rule declaration
  | SpliceD ...         -- Splice declaration
  | DocD ...            -- Documentation comment declaration
  | RoleAnnotD ...      -- Role annotation declaration
  | XHsDecl ...
```

[compiler/GHC/Hs/Binds.hs]

```
data HsBindLR idL idR
  = FunBind ...         -- Function-like Binding
  | PatBind ...         -- Pattern Binding
  | VarBind ...         -- Variable Binding
  | AbsBinds ...        -- Abstraction Bindings
  | PatSynBind ...      -- Patterns Synonym Binding
  | XHsBindsLR ...
```

[compiler/GHC/Hs/Expr.hs]

```
data HsExpr p
  = HsVar ...
  | HsUnboundVar ...
  | HsConLikeOut ...
  | HsRecFld ...
  | HsOverLabel ...
  | HsIPVar ...
  | HsOverLit ...
  | HsLit ...
  | HsLam ...
  | HsLamCase ...
  | HsApp ...
  | HsAppType ...
  | OpApp ...
  | NegApp ...
  | HsPar ...
  | SectionL ...
  | SectionR ...
  | ExplicitTuple
  | ExplicitSum
  | HsCase ...
  | HsIf ...
  | HsMultiIf ...
  | HsLet ...
  | HsDo ...
  | ExplicitList
  | RecordCon
  | RecordUpd
  | ExprWithTySig
  | ArithSeq
  :
```

An abstract syntax corresponding to Haskell user source.

Core language

[compiler/GHC/Core.hs]

```
type CoreProgram = [CoreBind]
type CoreBndr = Var
type CoreExpr = Expr CoreBndr
type CoreArg = Arg CoreBndr
type CoreBind = Bind CoreBndr
type CoreAlt = Alt CoreBndr

data Expr b
  = Var Id -- Variable
  | Lit Literal -- Literal
  | App (Expr b) (Arg b) -- Application
  | Lam b (Expr b) -- Lambda abstraction
  | Let (Bind b) (Expr b) -- Variable binding
  | Case (Expr b) b Type [Alt b] -- Pattern match
  | Cast (Expr b) Coercion -- Cast
  | Tick (Tickish Id) (Expr b) -- Internal note
  | Type Type -- Type
  | Coercion Coercion -- Coercion
```

A tiny explicitly-typed functional language.
Only ten data constructors based on System FC.

STG language

[compiler/GHC/Stg/Syntax.hs]

```
data GenStgTopBinding pass
  = StgTopLifted (GenStgBinding pass) | StgTopStringLit Id ByteString

data GenStgBinding pass
  = StgNonRec (BinderP pass) (GenStgRhs pass) | StgRec [(BinderP pass, GenStgRhs pass)]

data GenStgRhs pass
  = StgRhsClosure (XRhsClosure pass) CostCentreStack !UpdateFlag [BinderP pass] (GenStgExpr pass)
  | StgRhsCon      CostCentreStack DataCon [StgArg]

data GenStgExpr pass
  = StgApp      Id [StgArg]
  | StgLit      Literal
  | StgConApp   DataCon [StgArg] [Type]
  | StgOpApp    StgOp [StgArg] Type
  | StgLam      (NonEmpty (BinderP pass)) StgExpr
  | StgCase     (GenStgExpr pass) (BinderP pass) AltType [GenStgAlt pass]
  | StgLet      (XLet pass) (GenStgBinding pass) (GenStgExpr pass)
  | StgLetNoEscape (XLetNoEscape pass) (GenStgBinding pass) (GenStgExpr pass)
  | StgTick     (Tickish Id) (GenStgExpr pass)
```

A very small purely-functional language with the abstract machine (STG-machine) semantics.

Cmm language

[compiler/GHC/Cmm.hs]

```
type CmmProgram = [CmmGroup]
type CmmGroup   = GenCmmGroup CmmStatics CmmTopInfo CmmGraph
type CmmGraph   = GenCmmGraph CmmNode
```

[compiler/GHC/Cmm/Node.hs]

```
data CmmNode e x where
  CmmEntry ...                -- Entry
  CmmComment ...              -- Comment
  CmmTick ...                  -- Tick annotation
  CmmUnwind ...                -- Unwind pseudo-instruction
  CmmAssign :: !CmmReg -> !CmmExpr -> CmmNode O O -- Assign to register
  CmmStore ...                 -- Assign to memory location
  CmmUnsafeForeignCall ...     -- An unsafe foreign call
  CmmBranch ...                -- Goto another block
  CmmCondBranch ...            -- Conditional branch
  CmmSwitch ...                -- Switch
  CmmCall ...                  -- A native call or tail call
  CmmForeignCall ...           -- A safe foreign call
```

[compiler/GHC/Cmm/Expr.hs]

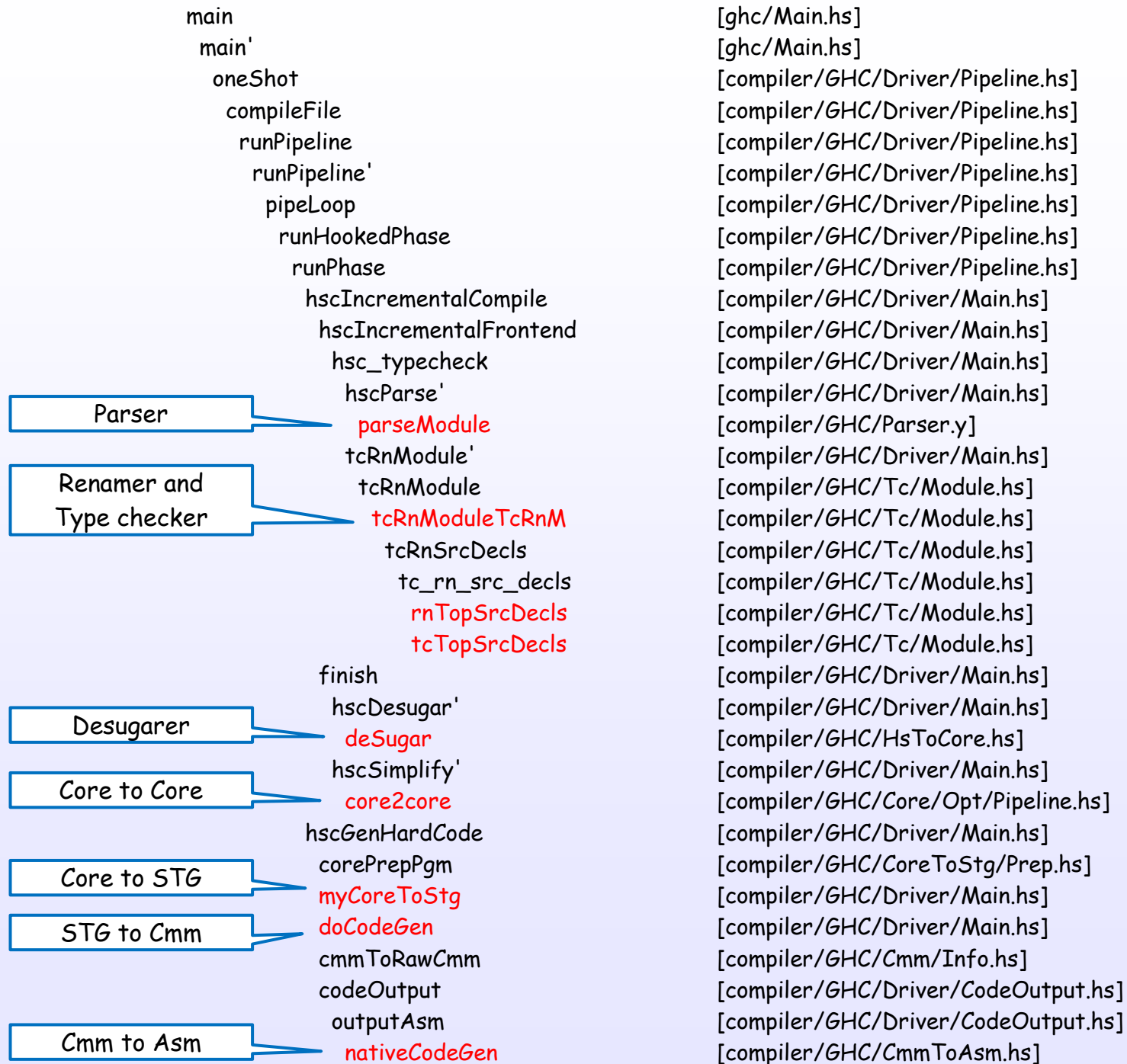
```
data CmmExpr
  = CmmLit      CmmLit           -- Literal
  | CmmLoad     !CmmExpr !CmmType -- Read memory location
  | CmmReg      !CmmReg          -- Contents of register
  | CmmMachOp   MachOp [CmmExpr] -- Machine operation (+, -, *, etc.)
  | CmmStackSlot Area {-# UNPACK #-} !Int
  | CmmRegOff   !CmmReg Int
```

A low-level imperative language with an explicit stack.

1. Compiler

Call graph

An example of a call graph



Appendix

Dump intermediate languages

Dump parser output:

```
$ ghc -ddump-parsed  
$ ghc -ddump-parsed-ast
```

Dump renamer output:

```
$ ghc -ddump-rn  
$ ghc -ddump-rn-ast
```

Dump type-checker output:

```
$ ghc -ddump-tc  
$ ghc -ddump-tc-ast
```

Dump Core:

```
$ ghc -ddump-ds  
$ ghc -ddump-simpl  
$ ghc -ddump-prep
```

Dump STG:

```
$ ghc -ddump-stg-final
```

Dump Cmm:

```
$ ghc -ddump-cmm  
$ ghc -ddump-opt-cmm
```

Dump asm/llvm:

```
$ ghc -ddump-asm  
$ ghc -ddump-llvm
```

Each intermediate language can be dumped using ghc's flags.

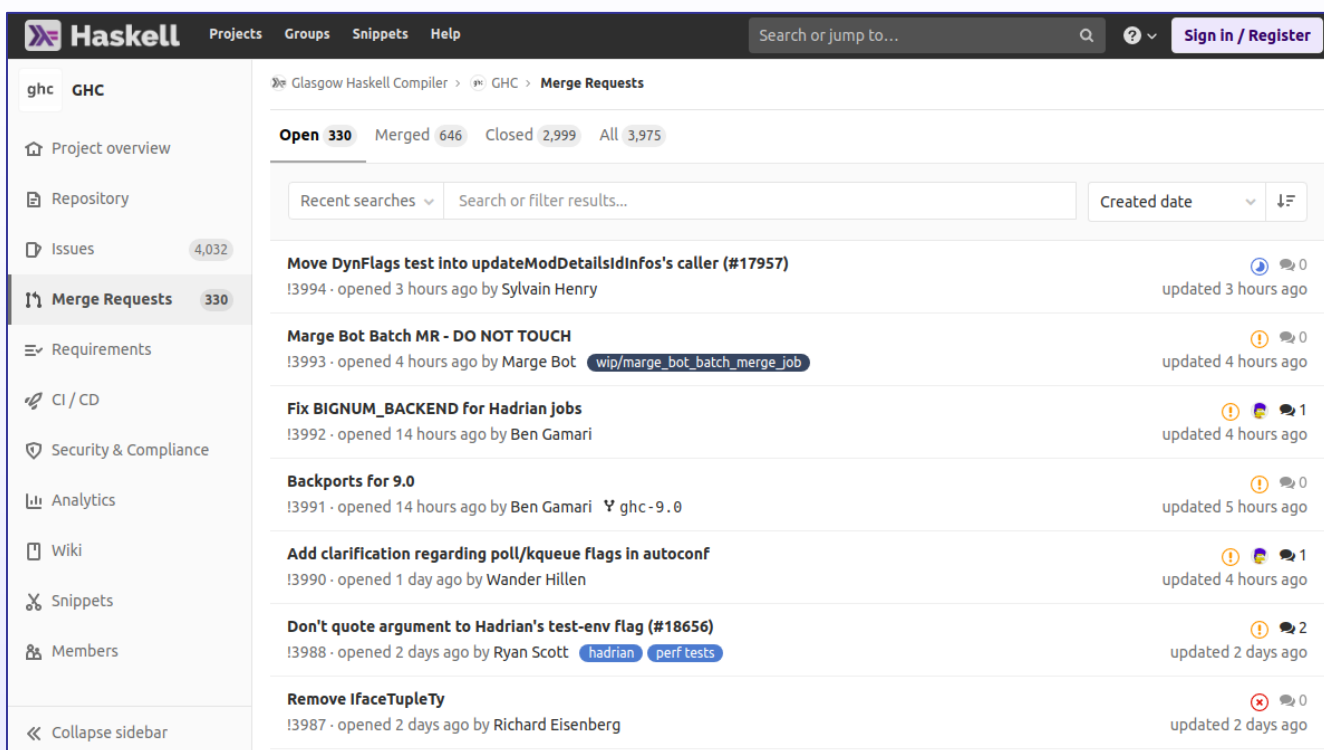
See the user's guide in detail :

https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/debugging.html#dumping-out-compiler-intermediate-structures

Patches are good entrances to dive into GHC

Merge Requests (Pull requests) :

https://gitlab.haskell.org/ghc/ghc/-/merge_requests



The screenshot shows the GitLab interface for the Haskell GHC project. The left sidebar lists various project sections, with 'Merge Requests' highlighted and showing 330 items. The main content area displays a list of merge requests for the Glasgow Haskell Compiler (GHC). The list includes the following items:

- Open 330** Merged 646 Closed 2,999 All 3,975
- Recent searches Search or filter results... Created date
- Move DynFlags test into updateModDetailsidInfos's caller (#17957)**
13994 · opened 3 hours ago by Sylvain Henry updated 3 hours ago
- Marge Bot Batch MR - DO NOT TOUCH**
13993 · opened 4 hours ago by Marge Bot [wip/marge_bot_batch_merge_job](#) updated 4 hours ago
- Fix BIGNUM_BACKEND for Hadrian jobs**
13992 · opened 14 hours ago by Ben Gamari updated 4 hours ago
- Backports for 9.0**
13991 · opened 14 hours ago by Ben Gamari [ghc-9.0](#) updated 5 hours ago
- Add clarification regarding poll/queue flags in autoconf**
13990 · opened 1 day ago by Wander Hillen updated 4 hours ago
- Don't quote argument to Hadrian's test-env flag (#18656)**
13988 · opened 2 days ago by Ryan Scott [hadrian](#) [perf tests](#) updated 2 days ago
- Remove IfaceTupleTy**
13987 · opened 2 days ago by Richard Eisenberg updated 2 days ago

The merge-requests page is a mine of practical codes.

References

References

Overview:

The Architecture of Open Source Applications: The Glasgow Haskell Compiler

<https://www.aosabook.org/en/ghc.html>

GHC Commentary: The Compiler

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler>

Compiling one module: *GHC.Driver.Main*

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/hsc-main>

A Haskell Compiler

<https://www.scs.stanford.edu/11au-cs240h/notes/ghc-slides.html>

Dive into GHC

https://www.stephendiehl.com/posts/ghc_01.html

Write a GHC extension in 30 minutes

<https://www.youtube.com/watch?v=bhhE2DxbrJM>

The GHC Commentary

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary>

The GHC reading list

<https://gitlab.haskell.org/ghc/ghc/-/wikis/reading-list>

References

Parser:

The Parser

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/parser>

Syntactic ambiguity resolution in the *GHC* parser

<https://blog.shaynefletcher.org/2020/04/syntactic-ambiguity-resolution-in-ghc.html>

The HsSyn types

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/hs-syn-type>

Renamer:

The renamer

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/renamer>

The Name type

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/name-type>

Type checker:

The *GHC* Commentary: Checking Types

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/type-checker>

The *GHC* reading list: Types and type inference

<https://gitlab.haskell.org/ghc/ghc/-/wikis/reading-list#types-and-type-inference>

References

Desugarer, Core:

Into the Core - Squeezing Haskell into Nine Constructors

https://www.youtube.com/watch?v=uR_VzYxvbxg

<https://www.erlang-factory.com/static/upload/media/1488806820775921euc2016intothecoresimonpeytonjones.pdf>

The Core type

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/core-syn-type>

Core-to-Core optimization pipeline

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/core-to-core-pipeline>

System FC, as implemented in GHC

<https://gitlab.haskell.org/ghc/ghc/blob/master/docs/core-spec/core-spec.pdf>

The GHC reading list: Optimisations

<https://gitlab.haskell.org/ghc/ghc/-/wikis/reading-list#optimisations>

Haskell to Core: Understanding Haskell Features Through Their Desugaring

<https://serokell.io/blog/haskell-to-core>

References

STG, Code generator:

Implementing lazy functional languages on stock hardware: the Spineless Tagless *G*-machine Version 2.5
<https://www.microsoft.com/en-us/research/wp-content/uploads/1992/04/spineless-tagless-gmachine.pdf>

Making a Fast Curry: Push/Enter vs. Eval/Apply for Higher-order Languages
<https://simonmar.github.io/bib/papers/eval-apply.pdf>

Faster Laziness Using Dynamic Pointer Tagging
<https://simonmar.github.io/bib/papers/ptr-tagging.pdf>

The STG syntax data types
<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/stg-syn-type>

I know kung fu: learning STG by example
<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/generated-code>

Overview of GHC's code generator
<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/code-gen/overview>

References

Cmm:

I know kung fu: learning STG by example

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/generated-code>

Cmm syntax

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/cmm-syntax>

cmm type [outdated]

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/cmm-type>

The C-- Language Specification Version 2.0

<https://www.cs.tufts.edu/~nr/c--/extern/man2.pdf>

Understanding the RealWorld

<https://www.well-typed.com/blog/95/>

Native/LLVM code generator:

Native Code Generator (NCG) [outdated]

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/backends/ngc>

The LLVM backend

<https://gitlab.haskell.org/ghc/ghc/-/wikis/commentary/compiler/backends/llvm>

Low Level Virtual Machine for Glasgow Haskell Compiler

<https://llvm.org/pubs/2009-10-TereiThesis.pdf>

Happy haskelling!