Calocom Compiler
编译器工程展示

x

xxx xxx xxx

https://github.com/permui/calocom

https://github.com/permui/calocom

CONTENTS

目录

语言特性

Language Features

Calocom 语言简述

Calocom 语言是什么?

Calocom 语言是一种函数式风格的托管编程语言具有:

- 代数数据结构
- 模式匹配
- 高阶函数
- 多维数组

等强大的特性。

Calocom 语言特性展示

Alergbic Data Type

和类型

```
data Nat = O
         | S(Nat);

data Weekday = Monday    | Tuesday | Wednesday
             | Thursday | Friday;
```

积类型

```
("", true, ())
(1, 2.3333, "hello, world")
```

代数数据类型

赋予语言强大的表现力

Calocom 语言特性展示

Pattern Matching

```
nat_is_equal: (a: Nat, b: Nat) -> bool {
    match (a, b) {
        (O, O) => true,
        (S(x), S(y)) => nat_is_equal(x, y),
        (_, _) => false
    }
}
```

全功能模式匹配

解构数据更容易

Calocom 语言特性展示

High-Order Function

```
add_x: (x: i32) -> i32 -> i32 {
    \ (y: i32) -> i32 => {
        x + y
    }
}

main: () -> () {
    let inc: i32 -> i32 = add_x(1);
    std.io.println(inc(1));
}
```

支持自动捕获自由变量的高阶函数

函数成为一等公民

Calocom 语言特性展示

Standard Library

```
import std.io
import std.array
import std.string
```

丰富的标准库函数

足以写完测试样例

Calocom 语言特性展示

Multiple Dimensions Array

```
complex_array: () -> [[i32]] {
    let arr: [[i32]] = std.array.new(
        5,
        \ (_: i32) -> [i32] =>
            std.array.new(
                5,
                \ (_: i32) -> i32 =>
                    0));

    for i in 0 .. 5 {
        io.println(i);
        arr[i][i] = i;
    };
    arr
}
```

多维（变长）数组

虽然看起来略费眼

Calocom 语言特性展示

Control Flow Statement

```
test_if: () -> i32 {
    let i: i32 = 0;
    if i != 1 {
        i + 2
    } else {
        if i != 3 {
            i + 4
        } else {
            i + 5
        }
    }
}
```

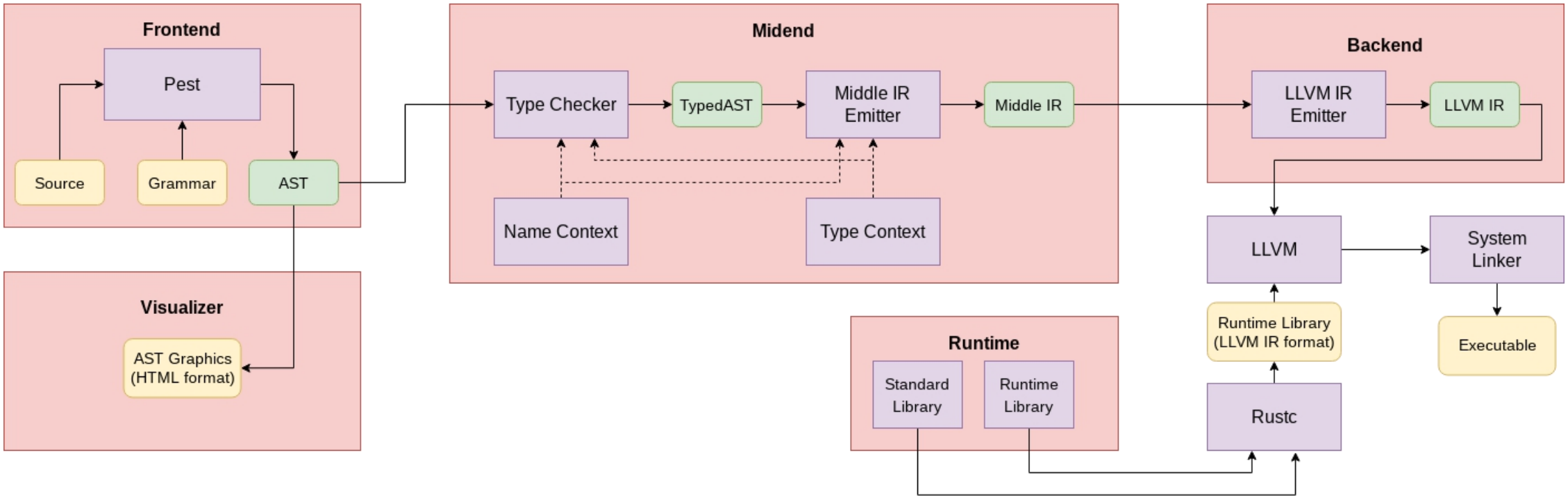
```
test_while: () -> i32 {
    let i: i32 = 0;
    while i < 100 {
        i = i + 1;
    };
    i
}

test_for: () -> i32 {
    let sum: i32 = 0;
    for i in 0 .. 10 {
        sum = sum + i;
    };
    sum
}
```

```
test_break_continue: () -> i32 {
    let i : i32 = 0;
    while true {
        if i > 10 {
            break;
        };
        if i == 0 {
            i = 9;
            continue;
        };
        i = i + 1;
    };
    i
}
```

常见的控制流语句

经典永流传

Calocom 编译器工作流程

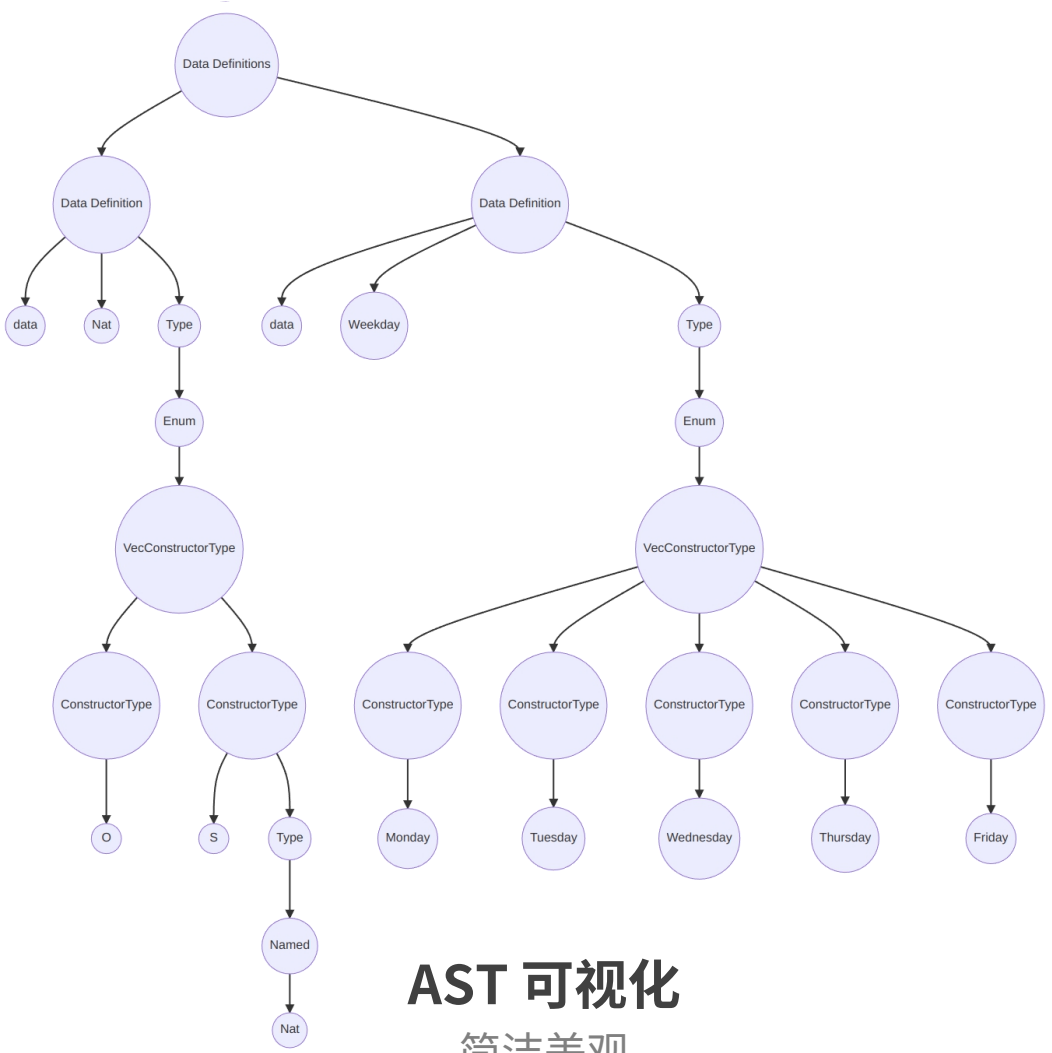


编译器前端

A Compiler Frontend is the Driver, Lexer & Parser

Calocom 编译器可视化

AST Visualizer



AST 可视化

简洁美观

Calocom 编译器前端

Compiler Driver

```
calocom-compiler 0.1.0

USAGE:
    calocom-compiler [OPTIONS] <FILE>

ARGS:
    <FILE>    Specify the input file

OPTIONS:
    -h, --help                    Print help information
    -l, --opt-level <LEVEL>       Specify the optimizing level [default: O3] [possible values: O0,
                                  O1, O2, O3]
        --llvm-pass-debug         Specify if need to enable debug log for llvm passes
    -o, --output <OUTPUT>         Specify the output file
    -r, --relocate <RELOCATE>     Specify the relocation type [default: pic] [possible values:
                                  static, dynamic-no-pic, pic, default]
        --runtime <RUNTIME>       Specify the runtime file [default: calocom_runtime.ll]
    -t, --type <TYPE>             Specify the type of the output file [possible values: llvm-bc,
                                  llvm-asm, mir, tast, ast, asm, bin, obj]
    -u, --visualize               Use this option to generate the visualized ast in html
    -V, --version                 Print version information
```

Driver 功能完备

用户友好型

Calocom 编译器前端

Compiler Lexer & Parser

- **Pest 是什么?**

Pest 是一种使用 PEG 文法的 Parser Generator

- **PEG 文法是什么?**

解析表达式（Parsing Expression Grammar, PEG）文法可以用来表示一个递归下降解析器

基于 Pest

站在巨人的肩膀的上

Calocom 编译器前端

Compiler Lexer & Parser

```
// auxiliary

WHITESPACE = _{ " " | "\t" | "\n" }
COMMENT = _{ "/*" ~ (!"*/" ~ ANY)* ~ "*/" | "//" ~ (!"\n" ~ ANY)* ~ "\n" }
```

辅助语法构件

分隔、空白与注释

Calocom 编译器前端

Compiler Lexer & Parser

```
// Basics

ident = @{ (LETTER | "_") ~ (LETTER | NUMBER | "_")* }

// literals

integer_lit = @{ ("+" | "-")? ~ ASCII_DIGIT+ }

float_lit = @{
    ("+" | "-")? ~ "." ~ ASCII_DIGIT+ ~ float_exp?
  | ("+" | "-")? ~ ASCII_DIGIT+ ~ "." ~ ASCII_DIGIT* ~ float_exp?
  | ("+" | "-")? ~ ASCII_DIGIT+ ~ float_exp
}
float_exp = @{
    ("e" | "E") ~ ("+" | "-")? ~ ASCII_DIGIT+
}

string_lit = ${ "\"" ~ string_lit_inner ~ "\"" }
string_lit_inner = @{ ("\\\\" | "\\\"" | (!"\"" ~ ANY))* }

bool_lit = @{ "true" | "false" }

Literal = { float_lit | integer_lit | string_lit | bool_lit }
```

基础语法构件

标识符和字面量

Calocom 编译器前端

Compiler Lexer & Parser

```
StatementBody = {
    Let
  | While
  | For
  | Return
  | Break
  | Continue
  | Asgn
  | Expression
}

Let = { "let" ~ VariableName ~ ":" ~ Type ~ "=" ~ Expression }

While = { "while" ~ Expression ~ BracketExpression }

For = { "for" ~ VariableName ~ "in" ~ ForRange ~ BracketExpression }

ForRange = { Expression ~ ".." ~ Expression } // 0..n means [0, n)

Return = { "return" }

Break = { "break" }

Continue = { "continue" }

Asgn = { Expression ~ "=" ~ Expression }
```

```
Expression = { Closure }

// closure

Closure = {
    "\\" ~ "(" ~ ParameterList ~ ")" ~ "->" ~ Type ~ "=>" ~ Closure
  | ControlFlow
}

// control flow expression

ControlFlow = {
    Match
  | If
  | Or
}

// match

Match = {
    "match" ~ Expression ~ "{" ~ MatchArms ~ "}"
}

MatchArms = { MatchArm ~ ("," ~ MatchArm)* }

MatchArm = { Pattern ~ "=>" ~ Expression }
```

语句和表达式语法构件

各种语句和表达式

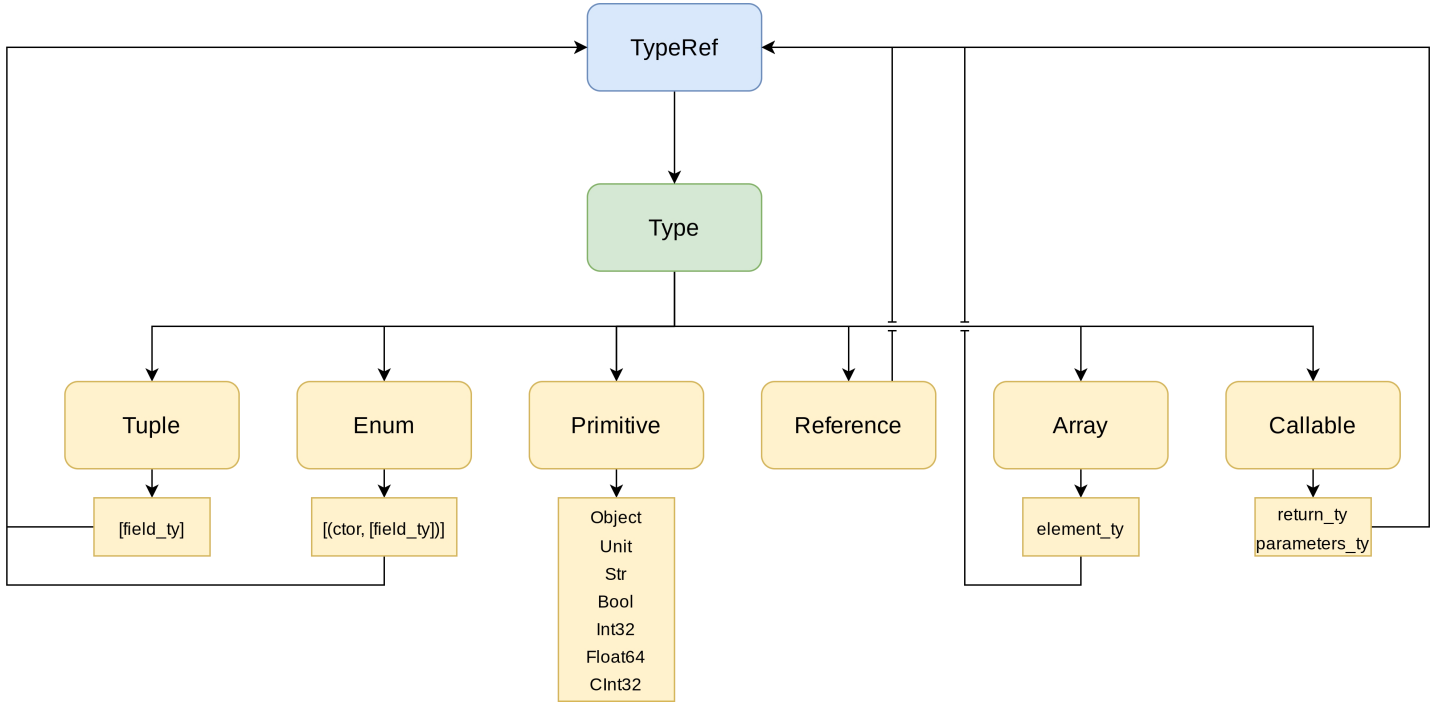
编译器中后端

Type Checker，Middle IR emitter & Codegen

Calocom 编译器中端

Types

```rust
#[derive(Debug, Clone, PartialEq, Eq, Hash)]
6 implementations
pub enum Type {
    Tuple {
        fields: Vec<TypeRef>,
    },
    Enum {
        name: String,
        ctors: Vec<(String, Vec<TypeRef>)>,
    },
    Primitive(Primitive),
    Opaque {
        alias: String,
    },
    Reference {
        refer: Either<TypeRef, String>,
    },
    Array(TypeRef),
    Callable {
        kind: CallKind,
        ret_type: TypeRef,
        parameters: Vec<TypeRef>,
    },
}
```



类型

Calocom 编译器中端

Type Context

```
#[derive(Debug, Clone)]
6 implementations
pub struct TypeContext {
    name_ref_map: HashMap<String, TypeRef>,
    type_ref_map: HashMap<Type, TypeRef>,
    prim_ref_map: [TypeRef; 7],
    ctor_map: Rc<RefCell<HashMap<String, TypeRef>>>,
    types: SlotMap<TypeRef, Type>,
}
```

- 名称 ↦ 类型引用
- 类型 ↦ 类型引用
- 原始类型的数值表示 ↦ 原始类型的类型引用
- 构造器名称 ↦ 构造器类型
- 类型引用 ↦ 类型

类型上下文

提供各种类型信息与操作

Calocom 编译器中端

Type Checker

1. 插入标准库函数的类型签名
2. 解析 import 列表
3. 解析 AST 类型（但存在不透明类型（只知道名字但不知道其具体是否存在的类型，用于类型的滞后声明和递归声明）
4. 完善类型环境，替换掉不透明类型
5. 检查各表达式、语句的类型，将类型信息附加到 AST 上
6. 形成 Typed AST

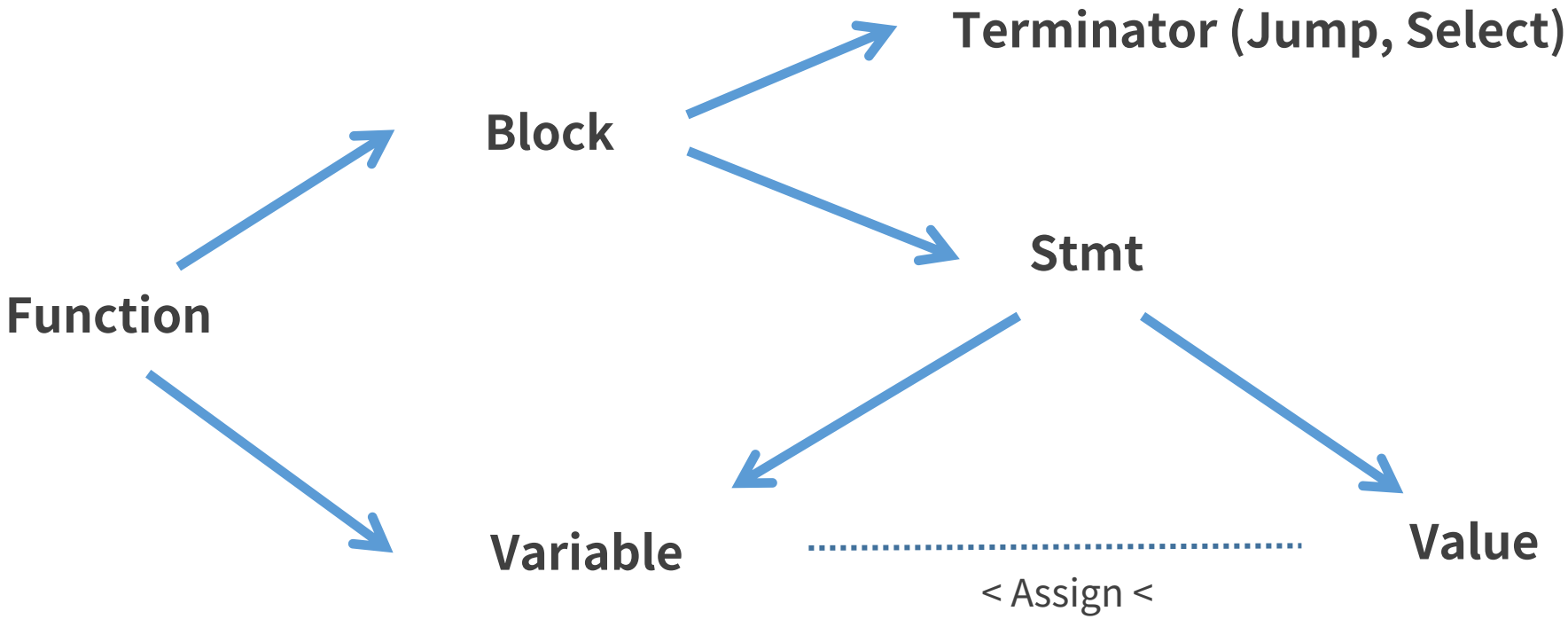
```
use crate::ast::BinOp::*;
if !match op {
    Or | And => {
        self.ty_ctx.is_boolean_testable_type(left.typ)
            && self.ty_ctx.is_boolean_testable_type(right.typ)
            && self.ty_ctx.is_type_compatible(left.typ, right.typ)
    }
    Eq | Ne => {
        self.ty_ctx.is_partially_equal_type(left.typ)
            && self.ty_ctx.is_partially_equal_type(right.typ)
            && self.ty_ctx.is_type_compatible(left.typ, right.typ)
    }
    Le | Ge | Lt | Gt => {
        self.ty_ctx.is_partially_ordered_type(left.typ)
            && self.ty_ctx.is_partially_ordered_type(right.typ)
            && self.ty_ctx.is_type_compatible(left.typ, right.typ)
    }
    Plus | Sub | Mul | Div | Mod => {
        self.ty_ctx.is_arithmetic_type(left.typ)
            && self.ty_ctx.is_arithmetic_type(right.typ)
            && self.ty_ctx.is_type_compatible(left.typ, right.typ)
    }
} {
    panic!("not compatible types for binary operator {:?}", op)
}
```

类型检查

得到定型后的抽象语法树

Calocom 编译器中端

Middle IR



中级 IR

解构控制流抽象到基本块级

Calocom 编译器中端

Middle IR Emitter

一个典型的函数体 MIR 生成流程

1. 设置新的名称环境（AST名称 ↦ {MIR名称，类型引用}）：
2. 依次创建返回值变量定义、参数变量定义、入口块、出口块、panic 块
3. 创建 FunctionBuilder 并初始化，用于保存生成状态，方便之后的函数体生成
4. 生成函数体
5. 收集函数体中的闭包定义所产生的函数体并一同返回至上层

中级 IR 生成器

生成 MIR

```rust
fn convert_fn_definition(&mut self, func: &TypedFuncDef) -> Vec<FuncDef> {
    let mut def = FuncDef {
        name: func.name.clone(),
        ..Default::default()
    };

    // setup the symbol table
    let mut var_ctx: NameContext<VarDefRef> = Default::default();

    // create the return value variable
    let ret_var = def.create_variable_definition_with_name(
        "ret.var".to_string(),
        func.ret_type,
        VariableKind::ReturnVariable,
    );

    let mut params_type = vec![];
    var_ctx.entry_scope();
    for TypedBind {
        with_at: _,
        var_name,
        typ,
    } in &func.params
    {
        let param = def.create_variable_definition_with_name(
            var_name.clone(),
            *typ,
            VariableKind::Parameter,
        );
        params_type.push(*typ);
        if var_name != "_" {
            var_ctx
                .insert_symbol(var_name.to_string(), param)
                .and_then(|_| -> Option<()> { panic!("parameter redefined") });
        }
    }

    def.initialize_blocks();
    let entry_block = def.entry_block;
    let mut fn_builder =
        FunctionBuilder::create(&mut def, entry_block, var_ctx, &self.name_ctx, &self.ty_ctx);

    // convert the function body
    fn_builder.build_function_body(&func.body, ret_var);

    // exit the symbol table scope
    fn_builder.var_ctx.exit_scope();

    let mut other_defs = fn_builder.anonymous_function;
    let mut def_vec = vec![def];
    def_vec.append(&mut other_defs);
    def_vec
}
```

Calocom 编译器中端

Middle IR Emitter

```rust
fn convert_fn_definition(&mut self, func: &TypedFuncDef) -> Vec<FuncDef> {
    let mut def = FuncDef {
        name: func.name.clone(),
        ..Default::default()
    };

    // setup the symbol table
    let mut var_ctx: NameContext<VarDefRef> = Default::default();

    // create the return value variable
    let ret_var = def.create_variable_definition_with_name(
        "ret.var".to_string(),
        func.ret_type,
        VariableKind::ReturnVariable,
    );

    let mut params_type = vec![];
    var_ctx.entry_scope();
    for TypedBind {
        with_at: _,
        var_name,
        typ,
    } in &func.params
    {
        let param = def.create_variable_definition_with_name(
            var_name.clone(),
            *typ,
            VariableKind::Parameter,
        );
        params_type.push(*typ);
        if var_name != "_" {
            var_ctx
                .insert_symbol(var_name.to_string(), param)
                .and_then(|_| -> Option<()> { panic!("parameter redefined") });
        }
    }

    def.initialize_blocks();
    let entry_block = def.entry_block;
    let mut fn_builder =
        FunctionBuilder::create(&mut def, entry_block, var_ctx, &self.name_ctx, &self.ty_ctx);

    // convert the function body
    fn_builder.build_function_body(&func.body, ret_var);

    // exit the symbol table scope
    fn_builder.var_ctx.exit_scope();

    let mut other_defs = fn_builder.anonymous_function;
    let mut def_vec = vec![def];
    def_vec.append(&mut other_defs);
    def_vec
}
```

自由变量分析（捕获深度法）

1. 每进入一个闭包体，就向 delimiter 栈压入当前的名称环境深度，将其称为 delimiter。反之离开闭包体则弹出一个 delimiter
2. 查找符号时，反向查找 delimiter 栈找到最大的小于符号所在深度的 delimiter，并记录对应 delimiter 距离，将之称为捕获深度。显然，绑定变量的捕获深度为 0，自由变量的捕获深度为一个正整数
3. 生成闭包环境时，我们利用变量的捕获深度决定对应变量是否需要在当前层捕获

中级 IR 生成器

生成 MIR

Calocom 编译器后端

Memory Layout Context

```
Type::Tuple { fields: x } => {
    let mut fields = vec![object_t.as_basic_type_enum()];
    for ty in x.iter() {
        let llvm_type = self.type_llvm_type_map.get(ty).unwrap();
        let field_ty = llvm_type
            .ptr_type(AddressSpace::Generic)
            .as_basic_type_enum();
        fields.push(field_ty);
    }
    self.type_llvm_type_map
        .get_mut(&ty_ref)
        .unwrap()
        .into_struct_type()
        .set_body(&fields[..], true);
}
Type::Enum { .. } => {
    self.type_llvm_type_map
        .get_mut(&ty_ref)
        .unwrap()
        .into_struct_type()
        .set_body(&[object_t, i8_p_t.into()], true);
}
Type::Primitive(_) | Type::Opaque { .. } => {
    unreachable!()
}
Type::Reference { refer } => {
    let referred_type = refer.as_ref().left().unwrap();
    let referred_type = self.type_llvm_type_map.get(referred_type).unwrap();
    self.type_llvm_type_map.insert(ty_ref, *referred_type);
}
```

内存布局上下文的初始化流程

（两趟生成任意一个 MIR 类型对应的 LLVM 类型）

1. 排除不透明类型、原始类型、非闭包可调用类型，为其他类型生成 LLVM 的 Opaque Struct 类型（这样在第二趟中所有的类型都已创建，于是可以产生指针递归的类型）
2. 排除不透明类型、原始类型、非闭包可调用类型，为其他类型设置其对应 LLVM 类型的 Struct 成员的类型

内存布局上下文

将 Calocom 类型映射到 LLVM 类型

Calocom 编译器后端

Name Mangling

例如：

main: () -> () 会被重整为：

_CALOCOM_PF_$4mainfCu

std.io.print: (object) -> () 会被重整为:

_CALOCOM_PF_3std2io5printfCuCo

这有效地防止了潜在的名称冲突

```
type ::= 'Co'   // Calocom.Object
type ::= 'Cu'   // Calocom.Unit
type ::= 'Cb'   // Calocom.Bool
type ::= 'Ci4'  // Calocom.Int32
type ::= 'Cs'   // Calocom.String
type ::= 'Cci4' // Calocom.CInt32
type ::= 'Cf8'  // Calocom.Float64

// Complex Type
type-list ::= 'l_' type* '_l'
type ::= 'Ce' context name                  // Enum
type ::= 'Ct' type-list                     // Tuple
type ::= 'Cr' type                          // Reference
type ::= 'Ca' type                          // Array
type ::= 'Clf' function-signature           // Callable (Function)
type ::= 'Clc' function-signature           // Callable (Closure)
type ::= 'Clt' function-signature           // Callable (Ctor)

function-signature ::= 'f' type type-list
generic-signature  ::= 'g' number-of-generic-params
generic-function-signature ::= function-signature generic-signature
specialization     ::= 's' type-list

// Context
context ::= name*   // Full restricted context
context ::= '$'     // Current context

polymorphic-function-name ::= '_CALOCOM_PF_' context name function-signature
specialized-function-name ::= '_CALOCOM_F_'  context name generic-function-signature specialization
```

名称重整

获取唯一的符号名

Calocom 编译器后端

Codegen

```rust
pub fn emit_all(&mut self, mir: &'a MiddleIR) {
    self.function_value_ctx.entry_scope();
    insert_library_function(
        &mut self.function_value_ctx,
        self.memory_layout_ctx.get_mut_mir_type_context(),
        self.module.as_ref(),
    );
    for func in mir.module.iter() {
        self.emit_fn_declaration(func);
    }
    for func in mir.module.iter() {
        self.emit_fn(func);
    }
    self.function_value_ctx.exit_scope();
    self.module.verify().unwrap_or_else(|msg| {
        eprintln!("{}", msg.to_string());
        panic!("verification failed")
    });
}
```

LLVM IR 生成流程

1. 插入标准库函数的 LLVM 声明
2. 生成所有函数的函数声明（便于递归调用以及滞后声明）
3. 完善所有函数的函数体
4. 验证 LLVM 模块（调用 LLVM 的 Verification Pass）

LLVM IR 生成器

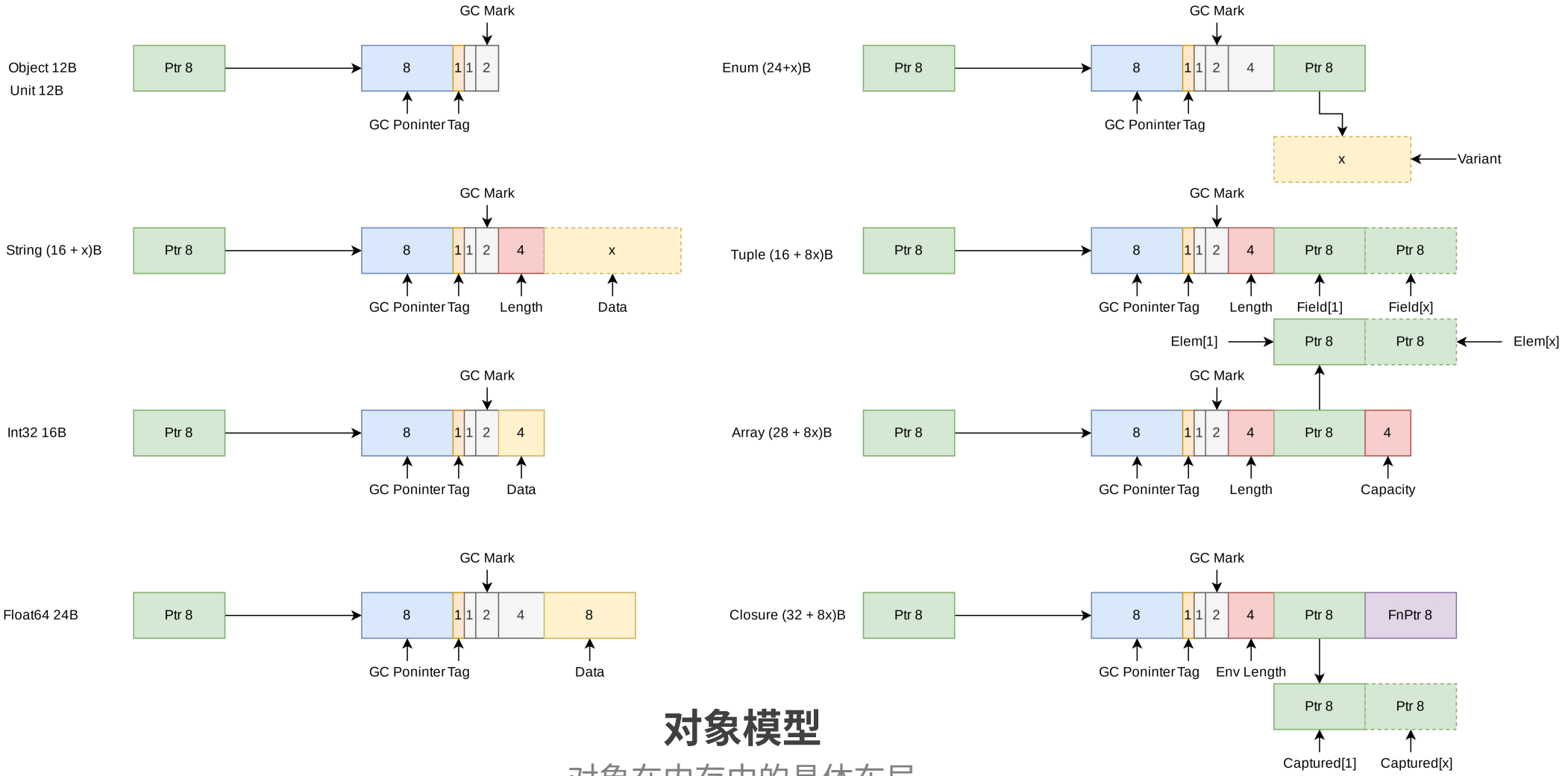
解构数据流操作，生成 LLVM IR

运行时环境

Runtime

Calocom 语言运行时环境

Object Model



对象模型

对象在内存中的具体布局

Calocom 语言运行时环境

Runtime Function

```rust
/// # Safety
///
/// This function should not be called directly by other crates
#[no_mangle]
#[export_name = "__calocom_runtime_array_index"]
pub unsafe extern "C" fn array_index(
    array: *mut _Array,
    index: *const _Int32,
) -> *mut *mut _Object {
    let subscript = extract_i32(index, 0);
    if subscript < 0 || subscript as u32 >= (*array).length {
        printf(
            const_cstr!("try access [%d] but limit is [%d]\n").as_ptr(),
            subscript,
            (*array).length,
        );
        panic(const_cstr!("array bound checking failed").as_ptr());
    }
    ((*array).data as *mut *mut _Object).add(subscript as usize)
}
```

示例 Runtime 函数：

带边界检查的数组访问

运行时函数

负责难以在编译器中实现的复杂任务

Calocom 语言运行时环境

Garbage Collection

垃圾回收

没来得及做

测试

Test

测试 1

Test 1

```
:: Write object file into "matrix-mul.o"
:: Try to link the object file
:: Write binary file into "matrix-mul"
fixed case 0 (size [1x1]x[1x1])...pass!
fixed case 1 (size [1x1]x[2x1])...pass!
fixed case 2 (size [1x4]x[4x1])...pass!
fixed case 3 (size [4x1]x[1x4])...pass!
fixed case 4 (size [1x25]x[25x1])...pass!
randomly generated case 0 (size [20x20]x[20x20])...pass!
randomly generated case 1 (size [20x20]x[20x20])...pass!
randomly generated case 2 (size [20x20]x[20x20])...pass!
randomly generated case 3 (size [20x20]x[20x20])...pass!
randomly generated case 4 (size [20x20]x[20x20])...pass!
randomly generated case 5 (size [20x20]x[20x20])...pass!
randomly generated case 6 (size [20x20]x[20x20])...pass!
randomly generated case 7 (size [20x20]x[20x20])...pass!
randomly generated case 8 (size [20x20]x[20x20])...pass!
randomly generated case 9 (size [20x20]x[20x20])...pass!
```

测试 2

Test 2

```
:: Write object file into "quicksort.o"
:: Try to link the object file
:: Write binary file into "quicksort"
fixed case 0 (size 0)...pass!
fixed case 1 (size 1)...pass!
fixed case 2 (size 2)...pass!
fixed case 3 (size 2)...pass!
fixed case 4 (size 3)...pass!
fixed case 5 (size 3)...pass!
fixed case 6 (size 3)...pass!
fixed case 7 (size 3)...pass!
fixed case 8 (size 3)...pass!
fixed case 9 (size 4)...pass!
fixed case 10 (size 9)...pass!
fixed case 11 (size 9)...pass!
fixed case 12 (size 10000)...pass!
fixed case 13 (size 10000)...pass!
fixed case 14 (size 4096)...pass!
randomly generated case 0 (size 10000)...pass!
randomly generated case 1 (size 10000)...pass!
randomly generated case 2 (size 10000)...pass!
randomly generated case 3 (size 10000)...pass!
randomly generated case 4 (size 10000)...pass!
randomly generated case 5 (size 10000)...pass!
randomly generated case 6 (size 10000)...pass!
randomly generated case 7 (size 10000)...pass!
randomly generated case 8 (size 10000)...pass!
randomly generated case 9 (size 10000)...pass!
```

测试 3

Test 3

```
:: Write object file into "advisor.o"
:: Try to link the object file
:: Write binary file into "advisor"
Test 0(fixed).....[Ok]
Test 1(fixed).....[Ok]
Test 2(fixed).....[Ok]
-----
Test 7(random).....[Ok]
Test 76(random).....[Ok]
Test 13(random).....[Ok]
Test 26(random).....[Ok]
Test 4(random).....[Ok]
Test 57(random).....[Ok]
Test 10(random).....[Ok]
Test 2(random).....[Ok]
Test 63(random).....[Ok]
Test 8(random).....[Ok]
Test 51(random).....[Ok]
Test 9(random).....[Ok]
Test 19(random).....[Ok]
Test 27(random).....[Ok]
Test 77(random).....[Ok]
Test 5(random).....[Ok]
Test 3(random).....[Ok]
Test 16(random).....[Ok]
Test 54(random).....[Ok]
Test 1(random).....[Ok]
Test 14(random).....[Ok]
Test 52(random).....[Ok]
```

THANK YOU

谢谢您的观看