OpenSC

- A Smart Contract Language

Linghan Kong (lk2811), Ruibin Ma (rm3708), Chong Hu (ch3467), Jun Sha (js5506), Rahul Sehrawat (rs3688)

OpenSC

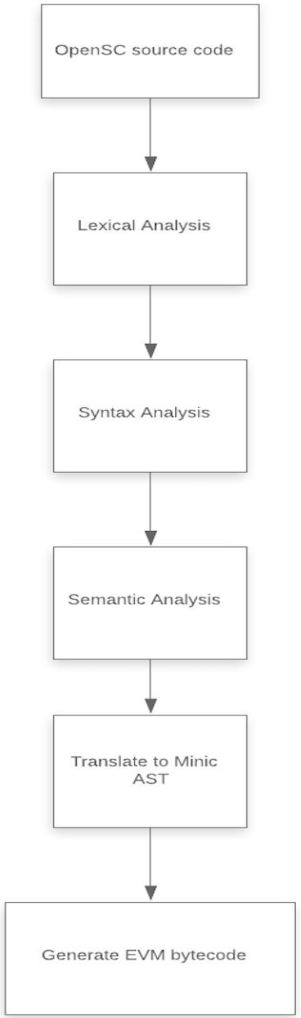
- A Smart Contract Language

- Introduction
- Architecture
- Lexical Analysis
- Syntax Analysis
- Semantic Analysis
- Translate to Minic
- Demo
- Future Work

Introduction

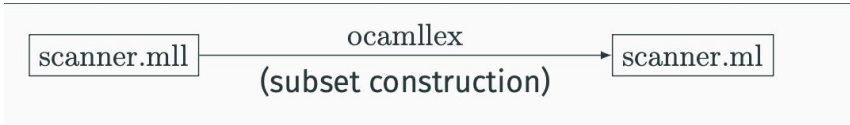
- Similar Languages: Scilla, Pact
- Uniqueness: State Transitions
- Inspiration: DeepSEA
- Smart Contract: Simplestorage, Auction, Token

Compiler Architecture



Lexical Analysis

- Comment
- General operation symbol
- Types
- Literal
- Built-in



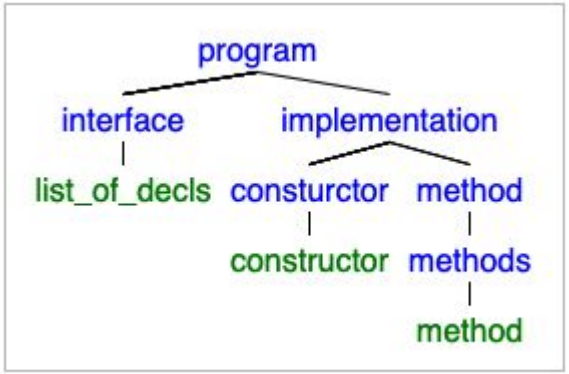

```
{open Parser}

let digits = ['0'-'9']
let letter = ['a'-'z' 'A'-'Z']

rule token = parse
  [' ' '\t' '\r' '\n']   { token lexbuf }
  |"/-"                 {multicomment lexbuf} (* multiple comment *)
  | '('                 { LPAREN }
  | ')'                 { RPAREN }
  | '{'                 { LBRACE }
  | '}'                 { RBRACE }
  | '['                 { LBRACK }
  | ']'                 { RBRACK }
  (*  General op *)
  | "=="                { EQ }
  | "!="                { NEQ }
```

Syntax Analysis

- Ocaml yacc
- CFG with attached semantic actions
- AST
  - op, type, expression, declaration,



OpenSC example

SimpleStorage:

Signature, constructor, methods

```
signature SimpleStorage {
    storage storedData : int;

    constructor c : (void) -> void;
    method get : () -> int;
    method set : (int) -> void;
}

constructor c (){
    storage
    returns void;
}

method get(){
    guard{}
    storage{}
    effects{}
    returns storedData;
}

method set(x: int) {
    guard{
        x > 0;
    }
    storage{
        storedData |-> x;
    }
    effects{}
    returns voidlit;
}
```

Semantic Analysis

Goal:

ast =========> sast

```
18  type expr =
19      | NumLit of int  (* number literal *)
20      | BoolLit of bool
21      | StrLit of string
22      | Id of string
23      | EnvLit of string * string
24      | Mapexpr of expr * expr list
25      | Binop of expr * op * expr
26      | Logexpr of expr * expr list
27      | Storageassign of expr * expr
28      | Comparsion of expr * op * expr
29      | Voidlit of string
```

```
5   type varscope =
6       | Sglobal
7       | Slocal
8
9   type sexpr = typ * sx
10  and sx =
11      | SNumLit of int  (* number literal *)
12      | SBoolLit of bool
13      | SStrLit of string
14      | SId of varscope * string
15      (* | SVar of sexpr * typ *)
16      | SEnvLit of string * string
17      | SMapexpr of sexpr * sexpr list
18      | SBinop of sexpr * op * sexpr
19      | SLogexpr of sexpr * sexpr list
20      | SStorageassign of sexpr * sexpr
21      | SComparsion of sexpr * op * sexpr
22      | SVoidlit of string
```

Semantic Analysis

- expr:
  - assign data type to expr based on each operation
  - check whether variable id in the symbol table; assign scope attribute
  - verify variable data types of binary operator, compare and assign expr
  - match map and event input data types with global variable declaration
  - match expr of method return with it return type
- constructor:
  - one and only one constructor in interface and implementation
- method:
  - match arguments data type with method declaration in interface
  - construct local symbol table

Semantic Analysis

Correct

```
/- correct -/
signature SimpleStorage {
    storage storedData : UInt;
    constructor c : UInt -> void;
    method set : (UInt) -> void;
}
```

✓

```
/- method set : (int, UInt) -> int; -/
/- correct -/
method set(x: int, y: UInt) {
  guard{}
  storage{}
  effects{}
  returns x;
}
```

✓

```
/- map balances : (Address) => UInt; -/
/- a : Address -/
/- correct -/
balances[a]
```

✓

Wrong

```
/- wrong -/
signature SimpleStorage {
    storage storedData : UInt;
    constructor c : UInt -> void;
    constructor c2 : UInt -> void;
    method set : (UInt) -> void;
}
```

✗

```
/- method set : (int, UInt) -> int; -/
/- wrong -/
method set(x: int, y: UInt) {
  guard{}
  storage{}
  effects{}
  returns y;
}
```

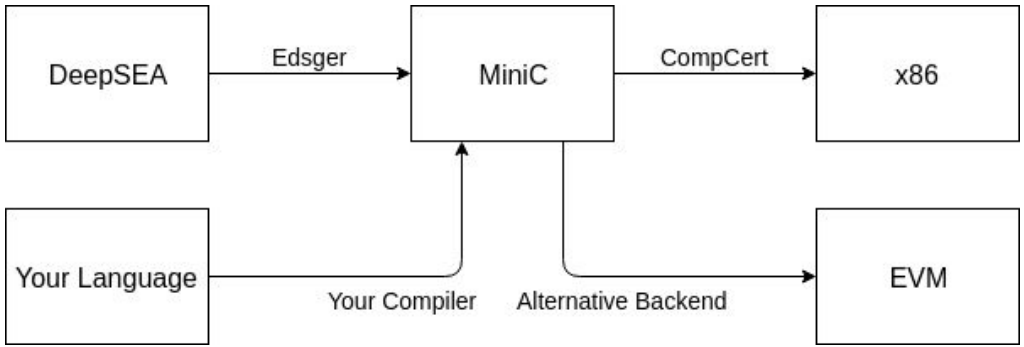
✗

```
/- map balances : (Address) => UInt; -/
/- a : Address -/
/- wrong -/
balances[ ]
/- b : int -/
balances[b]
```

✗

Translate to MiniC

- What is minic ?
  - The “IR”
  - Backend is ready
  - Just in AST format



Translate to MiniC

Goal:

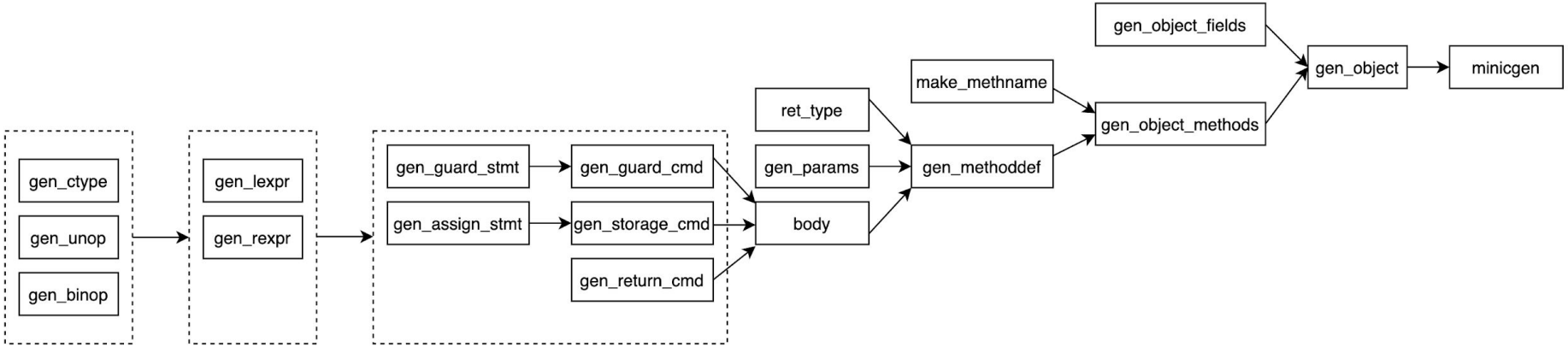
sast ==========> Minic AST

```
5   type varscope =
6       | Sglobal
7       | Slocal
8
9   type sexpr = typ * sx
10  and sx =
11      | SNumLit of int  (* number literal *)
12      | SBoolLit of bool
13      | SStrLit of string
14      | SId of varscope * string
15      (* | SVar of sexpr * typ *)
16      | SEnvLit of string * string
17      | SMapexpr of sexpr * sexpr list
18      | SBinop of sexpr * op * sexpr
19      | SLogexpr of sexpr * sexpr list
20      | SStorageassign of sexpr * sexpr
21      | SComparsion of sexpr * op * sexpr
22      | SVoidlit of string
```

```
9   type expr =
10  | Econst_int of Int.int * coq_type
11  | Econst_int256 of Int256.int * coq_type
12  | Evar of ident * coq_type
13  | Etempvar of ident * coq_type
14  | Ederef of expr * coq_type
15  | Eunop of unary_operation * expr * coq_type
16  | Ebinop of binary_operation * expr * expr * coq_type
17  | Efield of expr * ident * coq_type
18  | Earrayderef of expr * expr * coq_type
19  | Ehashderef of expr * expr * coq_type
20  | Ecall0 of builtin0 * coq_type
21  | Ecall1 of builtin1 * expr * coq_type
```

Translate to MiniC

- How to translate ?
  - Hierarchically



Translate to MiniC

- More Details of translation
  - ABI-compatible method id

Demo

Future Work

- Constructor
- Event and log
- Multi-key Mapping
- Multiple objects
- Control flow statement (if, for statement) “
    - Guard body is now translated into ‘if statement’

Acknowledgement

- Thanks to Professor Ronghui Gu, the instructor of our course, who brought us to the PLT world and let us realize the charm of functional programming and formal verification, both of which are what our project is based on.
- Thanks to Vilhelm Sjöberg, our project advisor, researcher at Yale and the primary creator of the DeepSEA project, who provided us with great information on everything about the DeepSEA project, and answered our many questions, which has been super helpful.
- Thanks to River Dillon Keefer and Amanda Liu, TAs of our course, who introduced the DeepSEA project to us and provided very inspiring and helpful ideas on the OpenSC language syntax among other project details.

Thank You!