

Dory – Software Design Document

Arun Olappamanna Vasudevan, Sonam Mandal, Udit Gupta

CSE 504.01 Compiler Design – Spring 2014: Class project: E-- compiler

Apr 16, 2014

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Change description** | **Date** | **Author(s)** |
| 0.1 | Initial draft | 11-Apr-2014 | Arun |
| 0.2 | System architecture | 16-Apr-2014 | Sonam, Udit, Arun |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[Revision History 2](#_Toc385385586)

[1 Introduction 4](#_Toc385385587)

[1.1 Purpose 4](#_Toc385385588)

[1.2 Scope 4](#_Toc385385589)

[1.3 Overview 4](#_Toc385385590)

[1.4 Reference Material 4](#_Toc385385591)

[1.5 Acronyms 4](#_Toc385385592)

[2 System Overview 4](#_Toc385385593)

[3 System Architecture 5](#_Toc385385594)

[3.1 Multi-pass Architecture 5](#_Toc385385595)

[3.2 Preprocessor 6](#_Toc385385596)

[3.3 Lexer and Parser 7](#_Toc385385597)

[3.4 Abstract Syntax Tree 7](#_Toc385385598)

[3.5 Type Checker 7](#_Toc385385599)

[3.6 Register assignments, Memory and stack allocation 7](#_Toc385385600)

[3.7 Intermediate Code Generation 8](#_Toc385385601)

[3.8 Optimizations 8](#_Toc385385602)

[4 Testing 8](#_Toc385385603)

[5 Effort Estimates and Plan 9](#_Toc385385604)

# Introduction

## Purpose

This software design document describes the architecture and system design of Dory, the class project: E-- compiler of course CSE 504.01 Compiler Design – Spring 2014. In addition to software design, the work effort estimates are also captured in the document.

The document will serve as reference for the team during later phases such as implementation and verification. It is also intended to be used for evaluation as part of course requirements.

## Scope

Dory is a compiler that converts E-- language to an abstract assembly language. The details of E-- compiler is available in the handouts provided for course.

Dory aims to generate an optimal assembly code without compromising on accuracy.

## Overview

Section 2 gives a modular overview of the system. This is followed by detailed module design in section 3.

Test design is explained in section 4.

Effort estimates and implementation plans are captured in section 5.

## Reference Material

|  |  |  |
| --- | --- | --- |
| **Document** | **Author** | **Version** |
| Compilers: Principles, Techniques, and Tools | Jeffrey Ullman, Ravi Sethi, and Monica S. Lam | 2 |
| <http://seclab.cs.sunysb.edu/sekar/cse504/#notes> | Prof. R. Sekar | - |

## Acronyms

|  |  |
| --- | --- |
| **Acronym** | **Expansion/Meaning** |
| AST | Abstract Syntax Tree |
| Gcc | GNU Compiler Collection |
| SDD | Software Design Document |

# System Overview

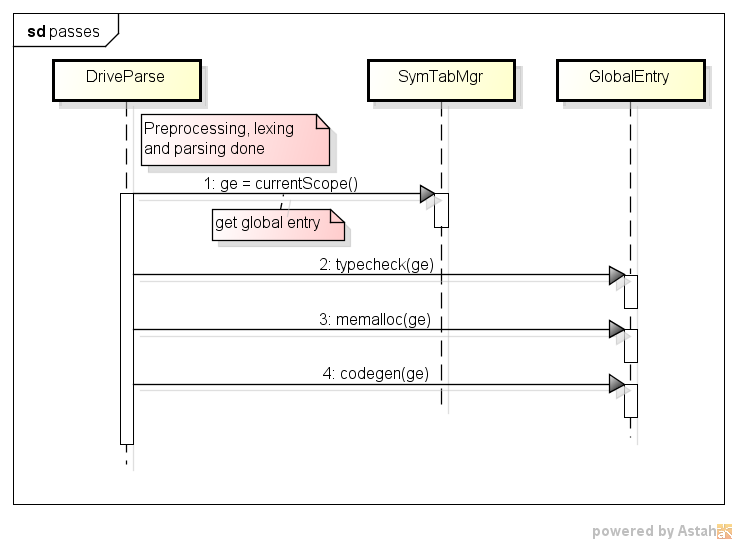
Below table lists modules of Dory.

|  |  |  |
| --- | --- | --- |
| **Module** | **Description** | **Remarks** |
| Preprocessor | Uses C/C++ preprocessor | Given |
| Lexer | Lexer for E-- compiler using flex | Given |
| Parser | Parser for E-- compiler using bison | Developed in previous assignments based on given class declarations |
| Abstract syntax tree | Generating abstract syntax tree for E-- text inputs | Developed in assignment 3 based on given class declarations |
| Symbol table | Scope stack containing symbol table entries | Developed in assignment 3 based on given class declarations |
| Type checker | E-- compiler supports a few pre-defined types as well as user defined types. Type checker makes sure typing errors, coercions, implicit, and explicit typings are handled |  |
| Register assignments, Memory and stack allocation | Allocations for static/global variables and local variables |  |
| Code generation | Generate assembly code |  |
| Optimizations | Optimizations are performed after data and control flow analyses |  |

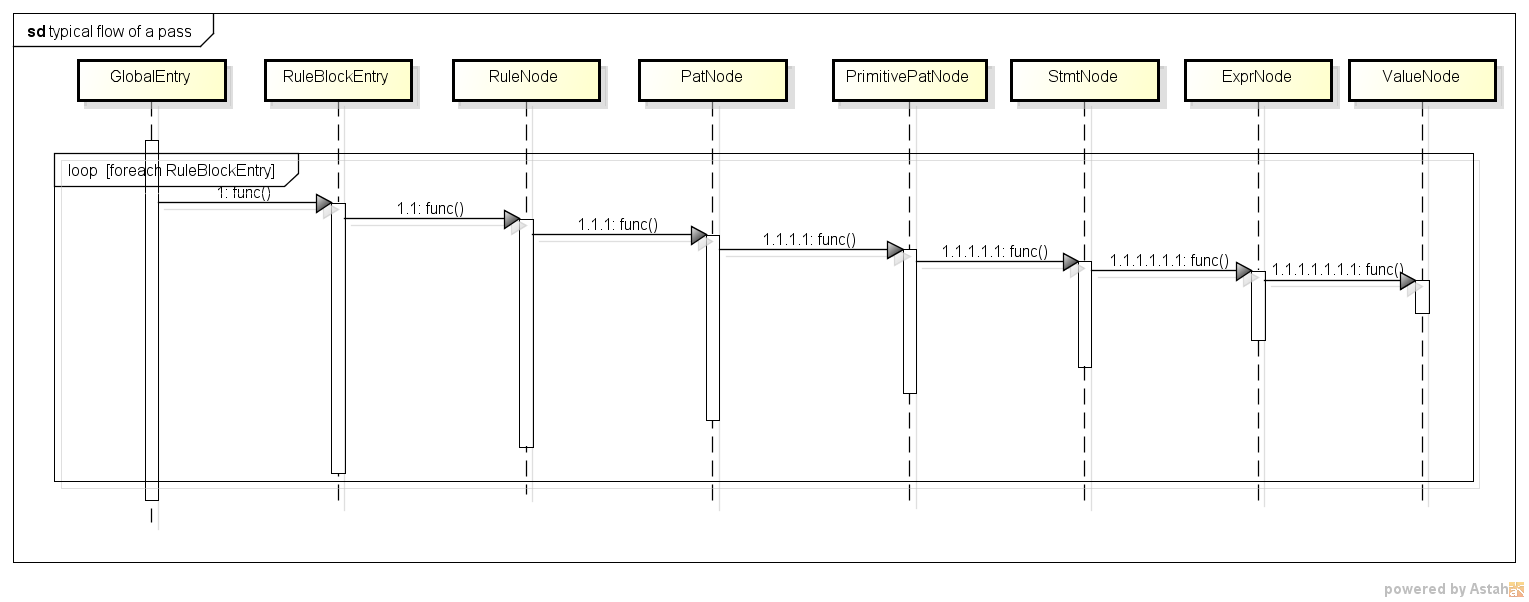
# System Architecture

## Multi-pass Architecture

Dory is a multi-pass compiler. The first pass takes code file as input and generates a preprocessed output. The second pass reads preprocessed file as input and generates symbol table entries and AST. This is done in previous assignments. Further passes work on AST and symbol table entry as shown below:



For each pass, the typical sequence of operations are as follows:



## Preprocessor

Removes comments and supports #includes and conditional compilation. Preprocessor of gcc is used.

## Lexer and Parser

Generated using flex and bison tools.

## Abstract Syntax Tree

The class declarations and relations between them are already provided. They are listed here for reference.

|  |  |
| --- | --- |
| **Class** | **Description** |
| ProgramElem | Program element – responsible for keeping line number and column number |
| AstNode | Abstraxt Syntax Tree base class |
| BasePatNode | Base class for pattern node |
| PrimitivePatNode | Primitive pattern node |
| PatNode | Pattern node |
| ExprNode | Expression node base |
| RefExprNode | Reference expression node |
| OpNode | Node for operations (binary and unary) |
| ValudeNode | Constant values in expressions |
| InvocationNode | Function call |
| RuleNode | Class for rules |
| StmtNode | Base class for statement |
| ExprStmtNode | Expression statement |
| CompoundStmtNode | Statements in compound block |
| IfNode | If-else statement |

## Type Checker

In this pass, type checking is performed

|  |  |  |
| --- | --- | --- |
| **Id** | **Class** | **Type** |
| 1 | RefExprNode | Look up in symbol table for type |
| 2 | OpNode | Coerce lower type to higher type in case of binary operations. Subtype checking and type coercions to be handled. |
| 3 | ValueNode | Type of value |
| 4 | InvocationNode | Match formal parameter types with actual parameter. Actual parameters must be subtypes. Coercion to be done if required. |
| 5 | IfNode | Condition must be of bool type and actual return type must be subtype of declared return type |
| 6 | RuleNode | Parameter matching for event |
| 7 | PrimitivePatNode, PatNode | Sequence operations checking for patterns |

## Register assignments, Memory and stack allocation

The strategy is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **Variable type** | **Strategy** | **Remarks** |
| 1 | Global variables | Assign in memory at offsets from global section |  |
| 2 | Local variables of other functions | Stack | We are thinking of using registers as well. |
| 3 | Temporary | Registers |  |

## Intermediate Code Generation

Code generation is done for:

|  |  |  |
| --- | --- | --- |
| **Id** | **Code gen for** | **Remarks** |
| 1 | Event |  |
| 2 | Function | Prologue and epilogue codes |
| 3 | Loop |  |
| 4 | Condition |  |
| 5 | Expression (logical) |  |
| 6 | Expression (arithmetic) | Floating point and integer arithmetic are to be handled differently |

## Optimizations

We plan to look into data and control flow analyses. The optimizations that can be performed once we have the graphs are mentioned below.

|  |  |  |
| --- | --- | --- |
| **Id** | **Category** | **Optimizations** |
| 1 | Data flow forward analysis (Reaching definition) | Constant propagation |
| 2 |  | Copy propagation |
| 3 | Data flow backward analysis (Live variable analysis) | Dead code elimination |
| 4 |  | Common sub-expression elimination |
| 5 | Control flow analysis | Remove unreachable code (Reachability) |
| 6 |  | Loop management |
| 7 | Others | Strength reduction |
| 8 |  | Jump threading |

# Testing

Unit testing (white box testing) is expected to be done by developer. A peer-level unit testing is also planned.

For black box testing, the test vectors and expected results are already provided. We plan to use script for test automation.

# Effort Estimates and Plan



Table below captures work distribution between team members. The way to read this table is as follows:

* For each column header, section number to refer is mentioned
* Ids mentioned under each column header correspond to table in the particular section

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Typecheck (3.5)** | **Memalloc (3.6)** | **Codegen (3.7)** | **Optimization[[1]](#footnote-1) (3.8)** |
| Arun | 5, 7 | 2 | 1, 3 | TBD |
| Sonam | 4, 6 | 1 | 2, 5 | TBD |
| Udit | 1, 2, 3 | 3 | 4, 6 | TBD |

1. Further and more concrete understanding needed to estimate and plan work on optimizations [↑](#footnote-ref-1)