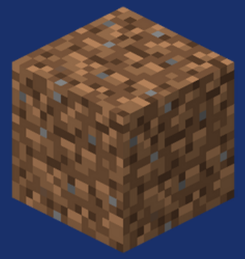
DRRTY 

Dylan Bamirny

Richard Lopez

Rania Alshafie

Trinity Sazo

Yes, we kept it (make) clean

Who Are We?

Dylan Bamirny
MANAGER

Richard Lopez
LANGUAGE GURU

Trinity Sazo
SYSTEM ARCHITECT

Rania Alshafie
TESTER

What is DRRTY?

What is DRRTY?

DRRTY is an imperative Python-like scripting language that offers back-end programmers a way to code front-end HTML pages.

Language Iterations

ORIGINAL

01

- Inspired by React
- Vision: create HTML using a Python/Java-like scripting language

REVAMP

02

- Discard HTML-type and opt for string output instead similar to printf()
- Discussion of using Gumbo as an HTML-linking library

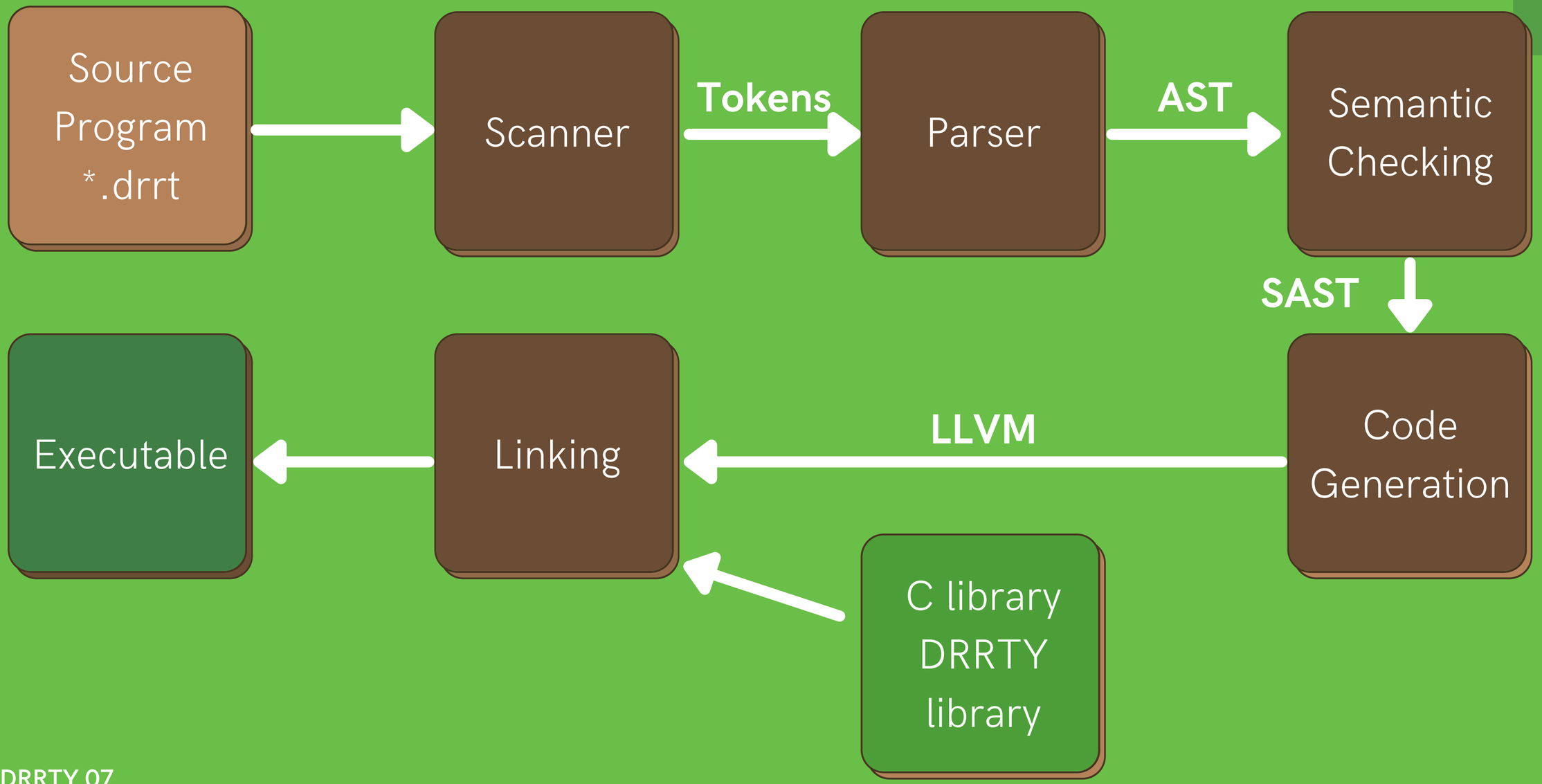
FINAL

03

- Discard Gumbo
- Created our own C library for linking
- Discard dictionaries, replace with just lists
- HTML built-in functions

Implementation

Architectural Design



KEY LANGUAGE FEATURES

- String implementation
- List implementation
- HTML built-in functions

Basic Syntax

- The DRRTY syntax follows Python mixed with Java/C.
- Brackets and semi-colons instead of indentation
- For basic data types, DRRTY supports int, float, list, bool, and str.
- For basic control flows, DRRTY supports for/while loops and if/else statements.

```
def int main(){
  makeHeader("Hello!");
  makeText("DRRTY's first program!");
  return 0;
}
```

Strings

```
str helloWorld;
helloWorld = "Hello, World!"
```

String Implementation

scanner.mll - tokenizes a string:

```
let unescape s =
    Scanf.sscanf ("\"" ^ s ^ "\"") "%S%!" (fun x -> x)
```

```
let string = '"' ( (ascii | escape)* as s) '"'
```

```
rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| "str"    { STRING }
| string             { STRING_LITERAL( (unescape s) ) }
```

str token points to a declaration of a string. A string literal is any sequence of zero or more characters enclosed in double quotes.

String Implementation

ast.ml - adds String type support for DRRTY:

```
type typ = Int | Bool | Float | Void | String | List of typ
```

```
type expr =
    Literal of int
  | Fliteral of string
  | BoolLit of bool
  | Id of string
  | StringLit of string
```

StringLit will carry the value of a string

sast.ml semantically checks AST

semant.ml returns semantically checked expr

String Implementation

drrtyparse.mly- parses str token:

```
%token FUNCTION END RETURN IF ELSE FOR WHILE INT BOOL FLOAT LIST VOID STRING
```

```
%token <string> ID FLIT STRING_LITERAL
```

```
typ:
    INT    { Int }
  | BOOL   { Bool }
  | FLOAT  { Float }
  | VOID   { Void }
  | STRING { String }
```

```
expr:
    LITERAL          { Literal($1)          }
  | FLIT             { Fliteral($1)         }
  | BLIT             { BoolLit($1)          }
  | STRING_LITERAL   { StringLit($1) }
```

Lists

```
list[str] names;
names = [“Dylan”, “Richard”, “Rania”, “Trinity”];

/* sets index 1 to "Rania" */
names.set(1, “Rania”);

/* get index 1 and print the string */
prints(names.get(1));
prints(names.get(2));
```

- get(), set(), add(), length()

List Implementation

listlibrary.c - creates list type and functions

```c
//Linked List Implementation

struct node{
    void * value;
    struct node * next;
};

struct list{
    int size;
    struct node * head;
};
```

```c
//Initialize list
struct list* list_init(){

    struct list *l;
    l = malloc(sizeof(struct list));

    if (l == NULL)
        return NULL;

    l->size = 0;
    l->head = 0;

    return l;
}
```

listlibrary.bc generated using clang; used in codegen to allow for struct datatypes and accompanying functions

List Implementation

codegen.ml - implements and links list type

```
let list_init_t = L.function_type list_t [||] in
let list_init_func = L.declare_function "list_init" list_init_t the_module in

let list_size_t = L.function_type i32_t [| list_t |] in
let list_size_func = L.declare_function "list_size" list_size_t the_module in

let list_get_t = L.function_type void_ptr_t [| list_t; i32_t |] in
let list_get_func = L.declare_function "list_get" list_get_t the_module in

let list_add_t = L.function_type i32_t [| list_t; void_ptr_t |] in
let list_add_func = L.declare_function "list_add" list_add_t the_module in
```

Following similar logic to Strings, drrtyparse.mly, ast.ml, sast.ml, semant.ml properly tokenize list elements, syntactically check the AST, and generate proper LLVM IR

HTML

```
def int main(){
  int i;
  int j;
  createHTMLDocument("printbig.css");
  for (i = 0; i < 100; i = i + 1){
      if( j == 0){
        createElement("div", "bigClass", "");
        j = 1;
      }else{
        createElement("div", "smallClass", "");
        j = 0;
      }
  }
  return 0;
}
```

- createHTMLDocument()
- createElement()
- makeHeader(), and more

HTML Implementation

htmllibrary.c- creates HTML functions:

```c
void createHTMLDocument(char *cssFile){
  printf("<!DOCTYPE html><html><head><link rel=\"stylesheet\" href=\"%s\"></head><body>", cssFile);
}

void createElement(char *elementName, char *className,  char *innerHtml){
  if( strcmp("img", elementName) == 0  || strcmp("input",elementName) == 0 ){
    printf("<%s class=\"%s\" src=\"%s\">\n", elementName, className, innerHtml);
  }else{
    printf("<%s class=\"%s\">%s</%s>\n", elementName, className, innerHtml, elementName);
  }
}
```

HTML Implementation

codegen.ml- implements and links HTML functions:

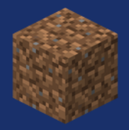
```
let createElement_t : L.lltype =
  L.function_type i32_t [| string_t; string_t; string_t |] in
let createElement_func : L.llvalue =
    L.declare_function "createElement" createElement_t the_module in

let createHTMLDocument_t : L.lltype =
  L.function_type i32_t [| string_t |] in
let createHTMLDocument_func : L.llvalue =
    L.declare_function "createHTMLDocument" createHTMLDocument_t the_module in
```

Built-in Functions

- createHTMLDocument()
- createElement()
- createHTML()
- makeHeader()
- makeText()
- makeImage()
- makeInput()

Testing



Test Suite

- Tests had a naming convention to indicate what test was run and what feature was being tested.
- Checks *.out against the output for test-*.drrt files, which are expected to pass

Test Plan

- Tests have been helpful in implementing our HTML functions and checking if a new feature is working properly without bugs.
- Tests are sample programs that a DRRTY user may write.

Test Example

```
def int main(){
    makeHeader("Different Types of Dirt");
    createHTML("<ul><li>Sand</li><li>Soil</
li><li>Thick Dirt</li><li>Dirt you  have on someone</
li><li>Skinny Dirt(Dust)</li></ul>");
    return 0;
}
```

Different Types of Dirt

- Sand
- Soil
- Thick Dirt
- Dirt you have on someone
- Skinny Dirt(Dust)

Future Work

1

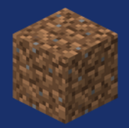
HTML TYPE/NESTED HTML

2

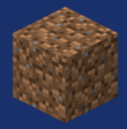
IMPLEMENT MORE ERROR CHECKS

3

ADD MORE DYNAMIC CSS SYTLING



DEMO



Questions?