EZ-ASCII

A Language for ASCII-Art Manipulation

Dmitriy Gromov
Joe Lee
Yilei Wang
Xin Ye
Feifei Zhong

December 18, 2012

Introduction

- **Purpose**: creating and manipulating ASCII Art

- **Goals**:
  - Easy creation, manipulation, and storage of ASCII images
  - Mapping / Converting intensities to characters
  - Flexible features with simple syntax

Tutorial : How to compile and run

1. Extract the EZ-ASCII compiler source files into a directory.
2. Run make to build the executable **ezac**.
3. The **ezac** executable takes a .eza source file as input, and allows some command-line parameters. A usage example is ezac [options] <source-file>.

Tutorial: A First EZ-ASCII Program

A simple "Hello, world!" program could look like the following:

```
d <- "Hello, world!";  // store string in variable d
d -> out;              // output d to stdout
```

Tutorial: More Examples

Looping:

```
for i <- 0 | i < 5 | i <- i + 1 {
  i -> out;
}
```

Functions:

```
d <- 2;
fun foo(p) {
    p -> "testfile.txt"; // output value of p to file
    d -> out;            // d refers to the global d, outputs 2
    a <- 4;              // a is a new local
    return a;
}
```

Types

- Usable Types
  - Int
  - Bool
  - String
  - Canvas
- Special Types
  - Void
- No Type Declarations
  - All can be inferred based on how they are created

Scoping

- Global scope
  - Global variables are accessible everywhere (but assignments only allowed in global scope)
- Local (function) scope
  - Local variables have lifetime within function body
  - Rules for variable lookup within a function:
    - Match against function parameter list
    - Match against global variables
    - Finally, create a new local variable
  - Rules imply that globals protected in function scopes

Modular Execution

- include “filename”;
  - Can include other files directly
  - Preprocessor replaces include with code from the given file
- Optional main
  - C style main function
  - Global statements executed first

Execution Storage

- Stack
  - Used to store references and integers
- “Heap”
  - Hashtable for complex types
    - String
    - Canvas
    - Bool
      - To not convert to 0 and 1
  - Lct (Load Complex Type) Byte code

Architecture – Primary Modules

Primary module	Function
ezac.ml	Top-level module with command-line options.
preprocess.ml	Recursively replace **include** statements with respective source files
scanner.mll	Converts source into stream of tokens
parser.mly	Parses stream of tokens into an AST tree (ast.ml)
ssanalyzer.ml	First pass through program, outputs compile-time errors (type errors, undefined var/fxn errors, etc…)
compiler.ml	Second pass through program, generate bytecode (bytecode.ml)
execute.ml	Executes bytecode

Architecture – Support Modules

Support module	Function
canvas.ml	Canvas type handling, operations, called during execution by execute.ml.
hashtypes.ml	Hashing support for complex (non-integer) types.
sast.ml	Define sast (semantically-checked ast)
ast.ml	Defines abstract syntax tree.
interpret.ml	Runs an AST program on the fly (no bytecode generation).
bytecode.ml	Defines byte-codes for compiler.

load_image.py – calls PIL for image loading
Makefile – builds
runtests.sh – shell script for unit testing

Canvas

- 2D Array of Integers
  - Each int -> Intensity
  - Domain -> [-1, Granularity)
- Rendering
  - Intensity Map -> {intensity, ascii-char}
  - If granularity < map cardinality
    - ~Evenly distribute intensity in map
  - -1 means null cell (not rendered)

Canvas

- Functions
  - Load : From jpeg, png or .i (intensity file)
  - Blank : New canvas with all 0s
  - Shift : Shift canvas up, left, down
  - Select : By point range(position), or boolean(intensity)
  - Set : 1 value to some selected range
  - Mask : Layer one canvas on top of another.

Canvas

- Load
  - PIL (Python Image Library)
    - Load image
    - Convert to grey scale
    - Shrink down to at most 100 x 100 image
      - If not square will keep ratio
      - If smaller than 100 x 100 stays same size
    - Stores intensity file to tmp dir in lang directory
    - File gets loaded into canvas
  - If .i file already loads file directly

Demo

Lessons Learned

1. OCaml has a steep learning curve but is effective for compilers.

2. GIT is of huge help in keeping track of the progress

3. Get AST, Scanner and Parser done early

4. Unit test ensures code work

5. Close Teamwork and frequent communication really important in this project

END

Thank You!