Columbia's AWK Replacement Language Demo

Darren Hakimi (dh2834)
Keir Lauritzen (kcl2143)
Leon Song (ls3233)
Guy Yardeni (gy2241)

May 8, 2017

Language Features

- Compiled AWK-like language used for text processing
  - *Pattern Action*
- .carl files compiled to LLVM
  - Linked with CARL library to form executable
- Supports regular expressions
  - //regex// delimiters
  - Ranges, concatenation, closure, choice
  - Regex is syntactically-checked at runtime
- Three types (like AWK):
  - Floats
  - Strings
  - Associative arrays (hash tables)
  - Void for functions
- Control flow (If, while, for)

Language Syntax

```
function type function_name(type formal){
    type local_var;
    return;
}
BEGIN { type global_var; }
//pattern// {action; }
END { print(global_var); }
```

C-like typed variables, formals, func.

Function variables are local

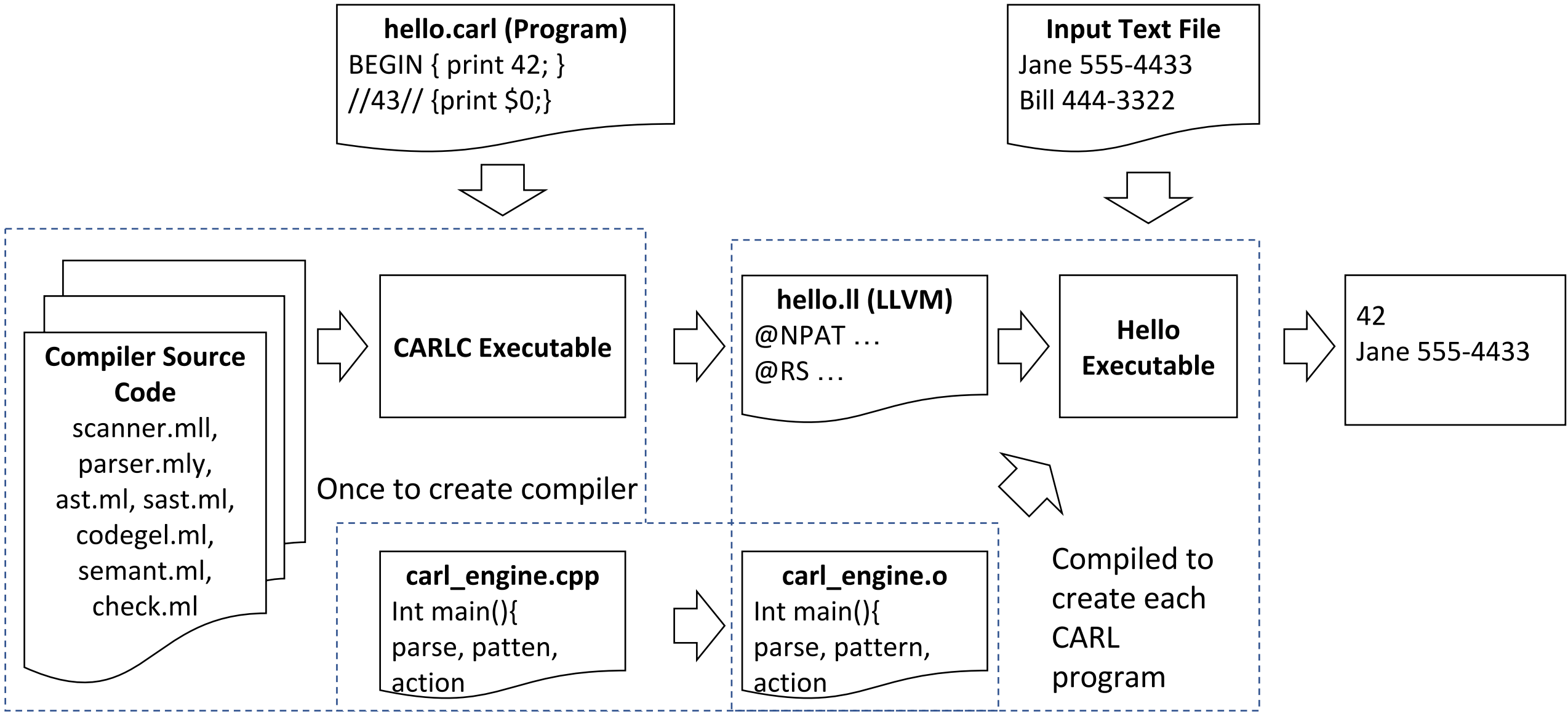
AWK's PATTERN-ACTION Syntax

AWK's Special Patterns: BEGIN / END

All variable are global

AWK's repeated processing on text inputs

Carl Engine Implementation



Carl Engine Implementation

carl_engine.cpp

```
int main(){
    opens the file;
    __init()
    parse using RS, FS
    for (int i; i < NPAT, i++)
    for each record {
        make_c_arrays(FS)
            regex(pattern[i])
            if regex.match
                (*action[i]) (int len,
char** cfields, int* cfl)
    }
    destroy_all()
    return 0;
}
```

carl_source.ll

```
int NPAT = 2;
char RS = '\n';
char FS = ' ';
char* patterns[NPAT];
void (*actions[NPAT])(int, char**,
int*);
void __init()
void destroy_all()
```

- *Externally CARL supports Float, String, Array, Void*
- *Internally CARL supports Float, Int, Int*, Char, Char*, Char**, (*Func.), (*Func.[]), Char*[], Void*
- *Functions and variables built at compile time in semant.ml (not externally visible)*

Array Implementation

- CARL arrays behave like Hashmaps
- We implemented arrays by creating wrappers around an existing C library and linking them with the CARL engine
- Wrapper functions:
  - create(): called by doing myArray = []
  - destroy(): called automatically by the engine at the end of every program
  - array_add_float(): myArray[“key”] = 42.0
  - array_add_string(): myArray[“key”] = “Thanks for all the fish”
  - array_retrieve_float(): float temp = myArray[“key”]
  - array_retrieve_string(): string temp = myArray[“Arthur Dent”]

https://github.com/petewarden/c_hashmap

Testing Approach

- Types of tests:
  - float, string, array tests
  - arithmetic operations
  - if/else statements
  - functions with each return type
  - regex
- Rundown of the testall script:
  - for each test file, run buildcarlp to create a executable file
  - run executable with input .txt file as argument
  - store output in .generated file
  - check if expected .out file matches with .generated and output .diff file
- Rundown of buildcarlp script
  - redirect .carl test file to carlc executable built by buildcarl and output the LLVM code to .ll file
  - convert .ll to assembly .s file
  - compile .s file and create .o file
  - compile wrapper.o file used with arrays, carl_engine.o and test .o file created in the prior step to form .test executable for running the test.

Demo 1

```
function float slight_increase(float val) {
    return val * 1.1;
}
BEGIN {
    float float_val1 = 4.321;
    string string_val1 = "I'm just a string.";
}
//float//  {
    while (float_val1) {
          print_float(float_val1);
          float_val1 = float_val1 - 1;
          float_val1 = slight_increase(float_val1);
    }
}
//string// {
    if (string_val1) {
          print_string(string_val1);
          while (string_val1) {
                print_string(string_val1);
                string_val1 = "";
          }
    }
}
END {
    if (float_val1 > 4.0) {
          print_string("strings are the best");
    } else {
          print_string("floats are the best");
    }
}
```

Demo 2

```
BEGIN {
   float temp_float;
   array_float myArrayFloat1 = [];
   array_float myArrayFloat2 = [];
   array_string myArrayString1 = [];
   string str = "key1";
   float val = 1;
   myArrayString1["abc"] = "2";
   myArrayString1["def"] = "MEANING OF LIFE?";
   myArrayFloat1[str] = val;
   myArrayFloat2[str] = myArrayFloat1[str] * 42;
}

END {
   string temp_string;
   temp_string = myArrayString1["abc"];
   print_string(temp_string);
   temp_string = myArrayString1["def"];
   print_string(temp_string);

   temp_float = myArrayFloat2[str];
   print_float(temp_float);
}
```

Demo 3

```
BEGIN {
   float a = 0;
   float b = 0;
   float c = 0;
   float d = 0;
   float e = 0;
   array_float myArray = [];
   string hiker = "key";
}
//Hitchhiker|Hitch Hiker// {
   a = a+1;
   myArray["Hitchhiker"] = a;
}
//Guide// {
   b = b+1;
   myArray["Guide"] = b;
}
//Galaxy// {
   c = c+1;
   myArray["Galaxy"] = c;
}
//Hitchhiker|Guide|Galaxy|Hitch Hiker//{
   d = d+1;
   myArray["Any"] = d;
}
//a*// {
   e = e+1;
   myArray["total"] = e;
}
END {
   print_string("Hitchhiker:");
   print_float(myArray["Hitchhiker"]);

   print_string("Guide:");
   print_float(myArray["Guide"]);

   print_string("Galaxy:");
   print_float(myArray["Galaxy"]);

   print_string("Any:");
   print_float(myArray["Any"]);

   print_string("total:");
   print_float(myArray["total"]);
}
```