MoveIt: The 2D Image Manipulation Language

by Benjamin Kornacki (blk2129)
Thomas Rantasa (tr2286)
Chengchen Sun(cs2890)

Motivation

- Graphics are often very tedious and bug-prone to implement
- No widely-used language that allows for easy implementation and modification
  - Significant background in language required
- Wanted to create language that allows 2D graphic manipulation without much programming background

Overview

0. Tutorial: How to use

1. Ast/Parser/Scanner

2. Compiler

3. Bytecode Interpreter

Tutorial: How to use

Syntax is similar to C

- execution begins from run(){ }
- data types
- functions

Tutorial: Data Types

There are 5 basic data types in MoveIt: int, string, line, ellipse, and rectangle

- an int is used primarily for coding and will never be printed to the display screen
- all other data types are structured as follows:
  - X:{ x_coordinate y_coordinate value... };
  - for an ellipse, x_coordinate and y_coordinate specify the center of the ellipse
  - for a line they specify the start point
  - for a rectangle and string the specify the bottom right corner
- value specifies either the height and width for a ellipse and rectangle, the string for a string, or the end point for a line

Tutorial: Operations

There are 2 main operations that can be performed to any string, line, ellipse, or rectangle

- -> : this operation move the object to a specified location
  - for example, if we were to execute the following:

    ellipse e;
    e = E:{ 100 200 50 60 };
    e -> { 200 25 };
  - then e will now have the value { 200 25 50 60 }
- => : this operation move the object by a specified distance
  - for example, if we were to execute the following:

    ellipse e;
    e = E:{ 100 200 50 60 };
    e => { 200 25 };
  - then e will now have the value { 300 225 50 60 }

Tutorial: Operations

There are 2 main operations that can be performed to any string, line, ellipse, or rectangle

- -> : this operation move the object to a specified location
  - for example, if we were to execute the following:

    ellipse e;
    e = E:{ 100 200 50 60 };
    e -> { 200 25 };
  - then e will now have the value { 200 25 50 60 }
- => : this operation move the object by a specified distance
  - for example, if we were to execute the following:

    ellipse e;
    e = E:{ 100 200 50 60 };
    e => { 200 25 };
  - then e will now have the value { 300 225 50 60 }

Tutorial: Print

- To print any non-integer object x to the window call print(x);
- however need to specify how long the objects should be printed for
- halt(time) will draw all print statements since the last halt to the window and will leave them there for time milliseconds

Demo

Pause for demonstration

Ast/parser/scanner

Ast: expressions: a single fully-implemented operation
statements: groups of expressions, with flow-control
vari_decl: structure for variable declaration
vari_value: structure to store variable value
func_decl: structure to store function declaration
program: list of vari_decl and func_decl to construct the whole program

Parser: construct ast using scanner and rules defined in parser.mly

Compiler

Takes the Abstract Syntax Tree and convert into bytecode list

- Id, Binop, Assign, Call, Moveto, Moveby, GetV,
- Flow control keywords: If, While, For, Return.

Handle Return, If, For, While, Ent.

- Carefully handling stack

Define ways to get global variable, local variable to index.

- int->2, string->30, shape->5 (but not allocating space)

Define built-in functions

- print, halt, print_int, dumpstack

Initialize whole program

- Sgraph, jump to run()

Bytecode

- General idea taken from MicroC and expanded/modified for MoveIt
- Stack more complicated due to sizes of structures
- Litint, Litsh, Litst
- Added Jsr(immediates)
- Moves
- Coordinates
- Other commands same concept as MicroC

Bytecode Interpreter

- General Idea from MicroC
- Stack much more difficult to implement
- Shapes identified by unique internal ID
- ID pushed last (information underneath ID)
- Allows quick position manipulation via sp-2 and sp-3
- Algorithms shape independent
- Significant drawback is String size limitation
    - restrictive and potentially wasteful
- Easy translation to OCaml Graphics
- Threads allow concurrency

...And more files to make it work

1. Toplevel tbc.ml
   Control what the ./tbc does when called

2. Makefile
   To include all the libraries and build options, and depency

...And test it!
Testcases to test from basic expressions and big demos

Lesson Learned

Start early
- helps manage work load
- more flexible with implementation

Start simple
- make sure you can run simple programs before trying to run complex ones

Plan
- on smaller projects "doing by coding" can work
- plan should be set in stone before first line is written
    - only allow potential for modification based on needs