MATHEW CARR

MSc. Project:

Code Generation Through Genetic Programming

Supervisor: David Jackson

Compiler

- Compilers translate from one language to another
- High level (human readable) to low level (efficiently executable)
- Almost all software is produced through use of a compiler

Why Consider Compilers?

- Compilers are not available for some architectures
- Where compilers are available, they may not take advantage of certain architecture-specific features:

  Advanced instructions: SIMD, DMA, etc.

Compilation Process

- Source code – High level program

  Parsing, lexical analysis

- Intermediate representation – Parse tree, symbol table.

  Code generation

- Object code – Low level program

  Linking

- Executable

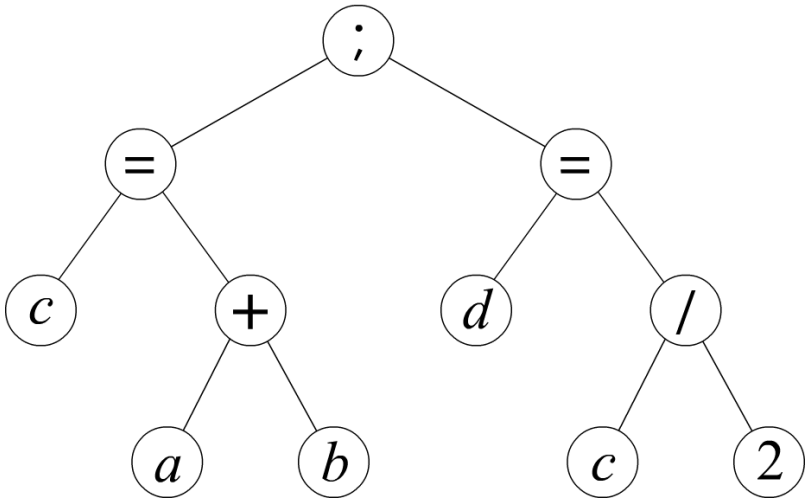
Code Generation

- Transformation from intermediate representation into object code
- Highly dependent on target architecture
- Register allocation
- Instruction selection
- Instruction sequencing

"Calculate the mean of the values of the variables a and b and store the result in variable d"

```
c = a + b;

d = c / 2
```




```
LOADS 0,    a    // load the value of variable a into r0
LOADS 1,    b    // load the value of variable b into r1
ADD   0, 0, 1    // add the values of r0 and r1; store result in r0
LOADV 1,    2    // load the direct value 2 into r1
DIVP  0, 0, 1    // divide value of r0 by that of r1; store result in r0
STORS 0,    d    // store the value of register 0 into variable d
HALT             // end program
```

Approach: Genetic Programming

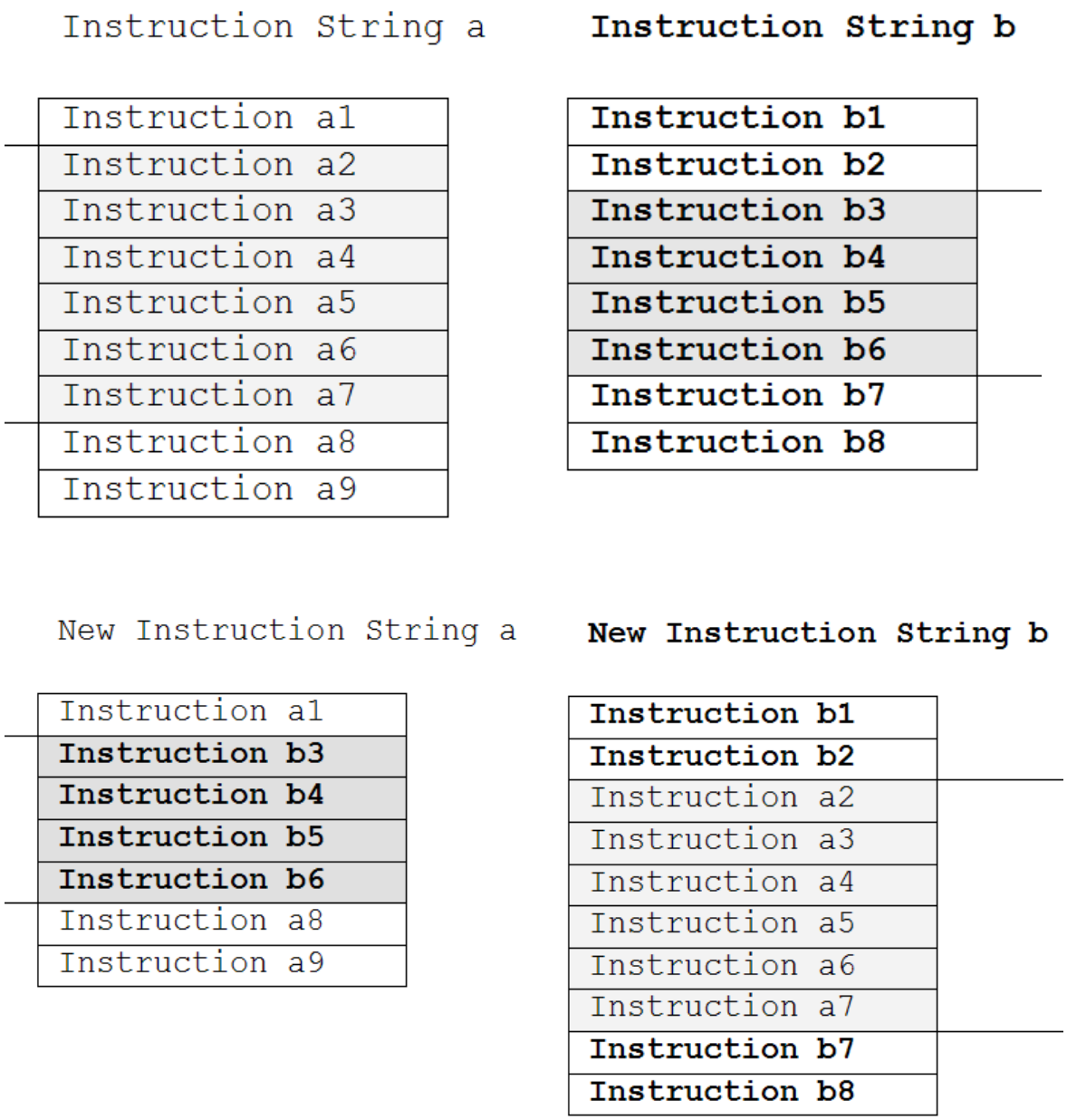
- Output computer programs through induction
- Automatically solves problems without having to know the size or shape of the solution in advance
- Requires method to determine suitability, or ‘fitness’, of candidate solutions

How Does GP Work?

- Creation of random initial population
- Fitness of all programs calculated
- New programs produced through recombination or alteration of ‘fit’ programs from the population
- Repeat until suitable program is found, or continue to find ‘fitter’ programs

Linear Genetic Programming

- We want to create programs consisting of a linear sequence of instructions
- Linear GP directly works upon, and returns, linear sequences of nodes
- Similar crossover and mutation operations


Instruction String a
Instruction a1
Instruction a2
Instruction a3
Instruction a4
Instruction a5
Instruction a6
Instruction a7
Instruction a8
Instruction a9
Instruction String b
Instruction b1
Instruction b2
Instruction b3
Instruction b4
Instruction b5
Instruction b6
Instruction b7
Instruction b8
New Instruction String a
Instruction a1
Instruction b3
Instruction b4
Instruction b5
Instruction b6
Instruction a8
Instruction a9
New Instruction String b
Instruction b1
Instruction b2
Instruction a2
Instruction a3
Instruction a4
Instruction a5
Instruction a6
Instruction a7
Instruction b7
Instruction b8

Mathew Carr

Project Approach

- Induce low level assembly-like programs for simple register machine architecture

- Specify architecture
- Determine method to calculate fitness of candidate solutions
- Determine method to calculate if candidate is correct

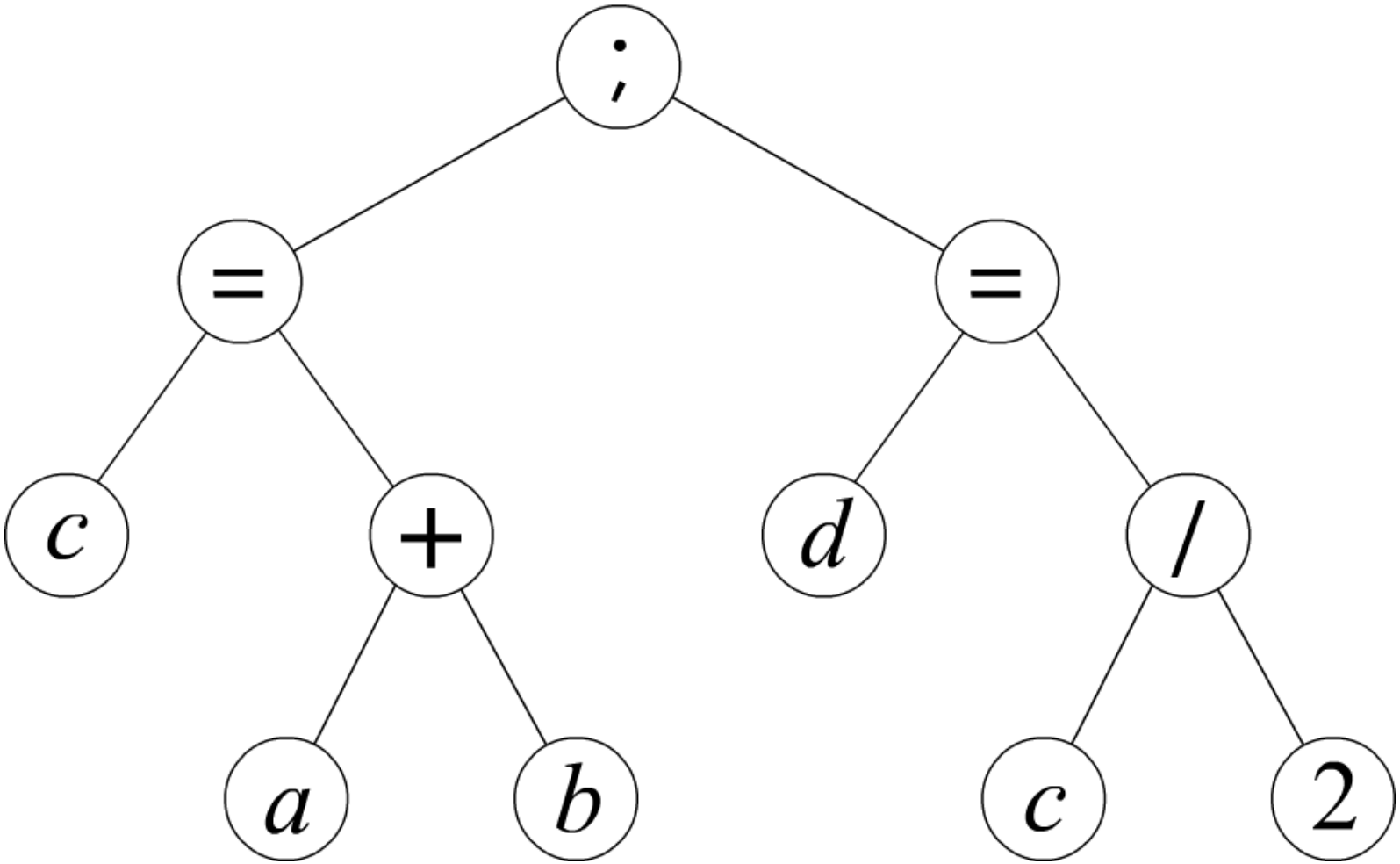
Fitness Cases

- Purely symbolic representations of system state are hard to work with
- Instead, work with sufficiently representative sampling of all possible input states

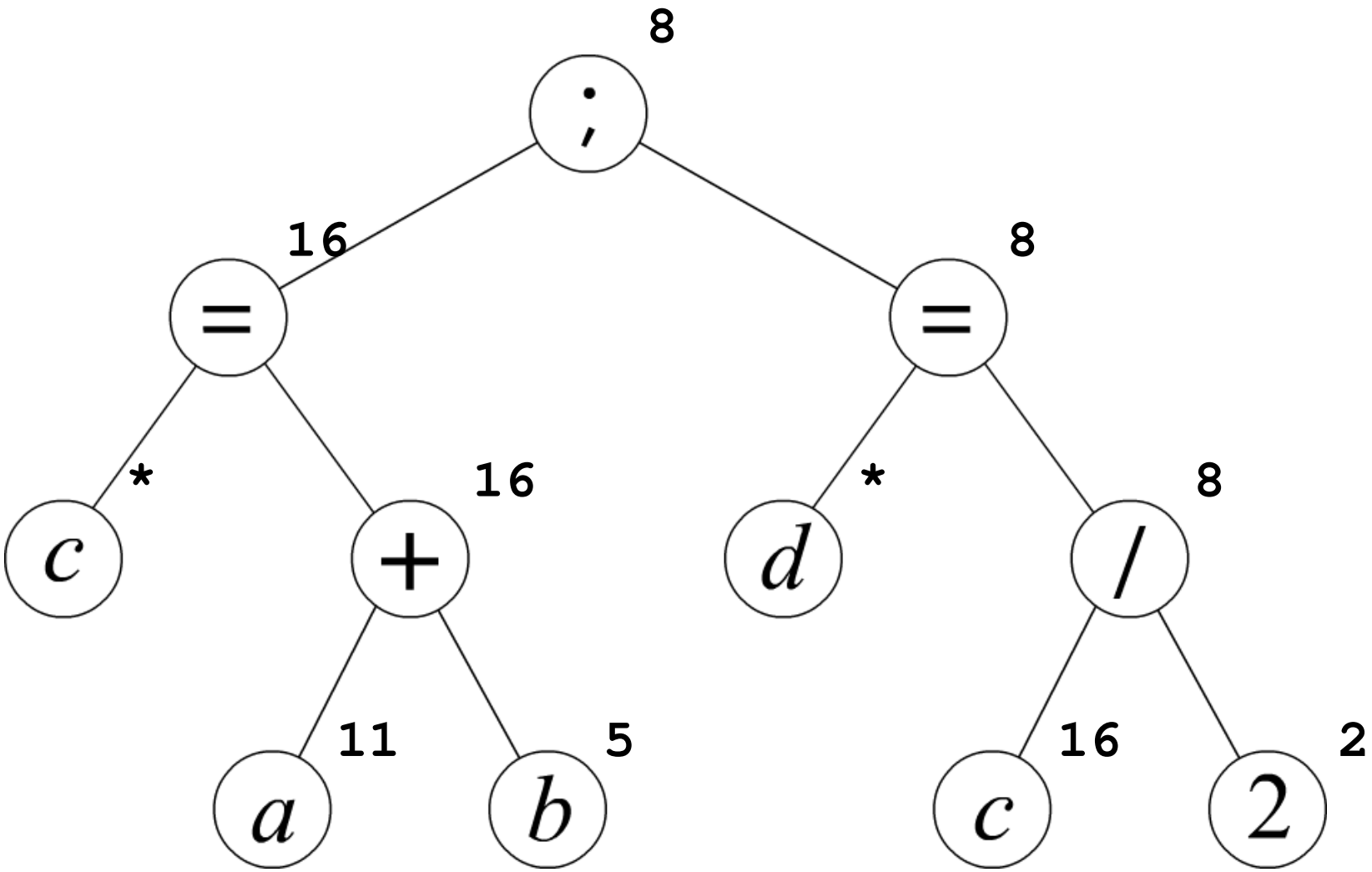
Fitness Evaluation

- Provide a real valued measure of how ‘close’ a program is to a solution
- Consider the actions taken within the system during execution
- Reward actions that seem productive; penalise actions that seem counterproductive
- Solution criteria are derived from contents of symbol table

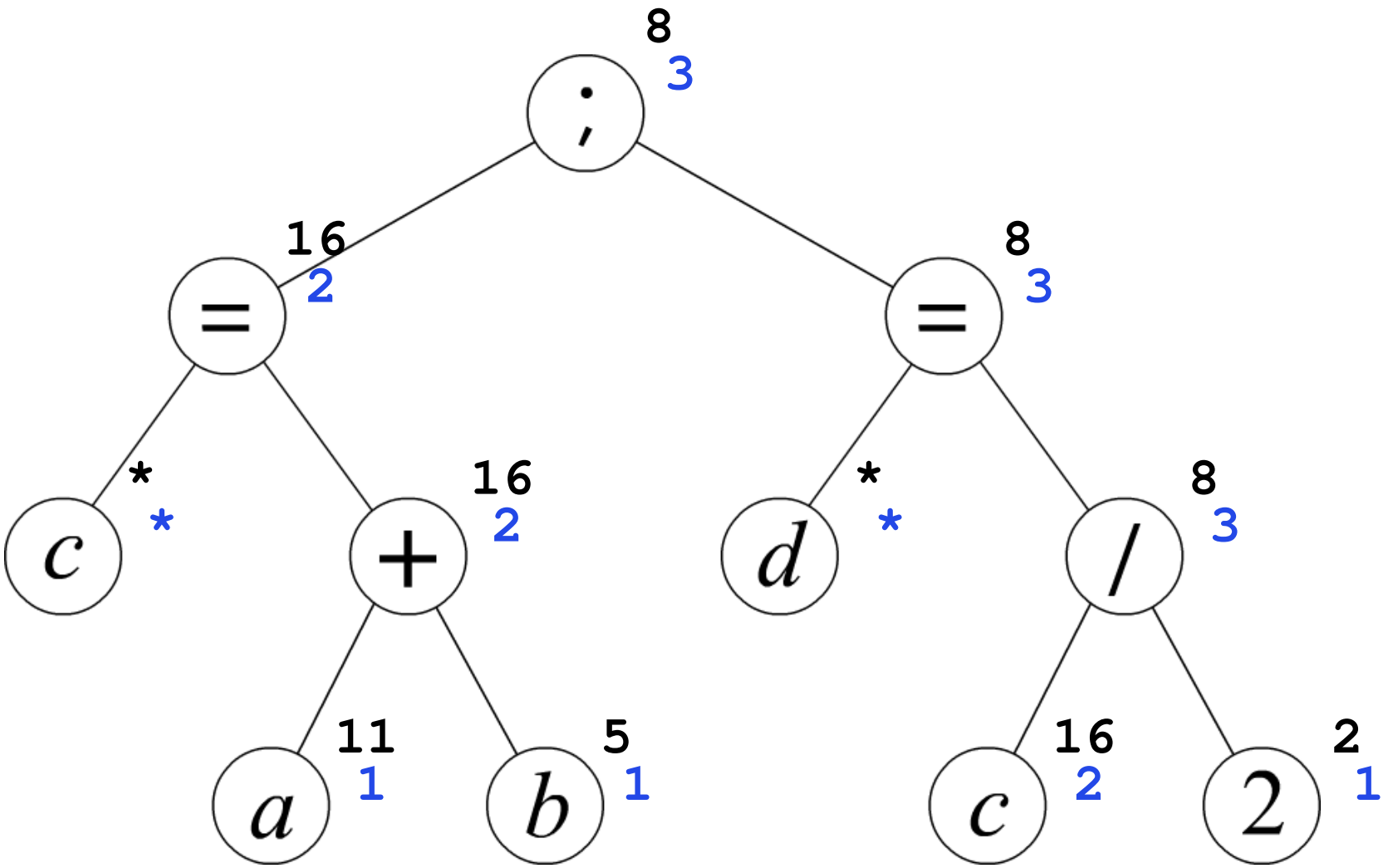
For all a and b:



For the case where a = 11 and b = 5:



For the case where a = 11 and b = 5:



For the case where a = 11 and b = 5:

Program is correct if program terminates and:
The value of d is 8;
The value of a is 11;
The value of b is 5

`LOADS 0, a`	r0 = a	r0 = 11	
`LOADS 1, c`	r1 = c	r1 = ?	
`ADD 0, 1, 0`	r0 = r1 + r0	r0 = ? + 11	r0 = ?
`LOADS 2, b`	r2 = b	r2 = 5	
`LOADS 0, a`	r0 = a	r0 =11	
`ADD 1, 2, 0`	r1 = r2 + r0	r1 = 5 + 11	r1 = 16
`LOADV 0, 2`	r0 = 2		
`STORS 2, d`	d = r2	d = 5	
`DIV 0, 1, 0`	r0 = r1 / r0	r0 = 16 / 2	r0 = 8

For the case where a = 11 and b = 5:

Program is correct if program terminates and:
The value of d is 8;
The value of a is 11;
The value of b is 5

`LOADS 0, a`	r0 = a	r0 = 11	
`LOADS 1, c`	r1 = c	r1 = ?	
`ADD 0, 1, 0`	r0 = r1 + r0	r0 = ? + 11	r0 = ?
`LOADS 2, b`	r2 = b	r2 = 5	
`LOADS 0, a`	r0 = a	r0 =11	
`ADD 1, 2, 0`	r1 = r2 + r0	r1 = 5 + 11	r1 = 16
`LOADV 0, 2`	r0 = 2		
`STORS 2, d`	d = r2	d = 5	
`DIV 0, 1, 0`	r0 = r1 / r0	r0 = 16 / 2	r0 = 8

`LOADS 0, a`	r0 = a	r0 = 11	
`LOADS 1, c`	r1 = c	r1 = ?	
`ADD 0, 1, 0`	r0 = r1 + r0	r0 = ? + 11	r0 = ?
`LOADS 2, b`	r2 = b	r2 = 5	
`LOADS 0, a`	r0 = a	r0 =11	
`ADD 1, 2, 0`	r1 = r2 + r0	r1 = 5 + 11	r1 = 16
`LOADV 0, 2`	r0 = 2		
`STORS 1, d`	d = r1	d = 16	
`LOADS 2, d`	r2 = d	r2 = 16	
`DIV 2, 2, 0`	r2 = r2 / r0	r2 = 16 / 2	r2 = 8
`STORS 2, d`	d = r2	d = 8	
`DIV 0, 1, 0`	r0 = r1 / r0	r0 = 16 / 2	r0 = 8

This appears to be a viable solution program

Does it work for all fitness cases?

If so, return it as the solution

Thank you

Any questions?