Petite VM

컴퓨터의 개념 및 실습 Project 3

TA. Juneyoung Park
juneyoung.park@snu.ac.kr

May 18, 2023
Due date: June 9, 2023

컴퓨터는 입력을 받아, 미리 정해진 규칙(프로그램)에 따라 행동한 후, 출력을 돌려주는 장치입니다. 컴퓨터가 다루는 정보는 크게 두 가지로 나눌 수 있습니다; 외부와 교환하는 정보(I/O)와 컴퓨터의 내부 상태(state, memory)입니다. 컴퓨터를 흉내내는 소프트웨어를 가상머신(virtual machine, VM)이라 하는데 이번 프로젝트에서는 Petite-Language (Plang)를 읽고 동작하는 VM을 Python으로 구현해 봅시다.

Objectives

I. 중위 표현식을 후위 표현식으로 고쳐주는 'in2post'를 작성하십시오. - `utils.py` **(30 pt.)**

II. Plang VM, `PTVM`**을 구현하십시오. -** `petite_vm.py`**,** `exceptions.py` **(60 pt.)**

(Bonus) 고정 비트 길이 정수형 `ixx`**를 구현하십시오. -** `ixx.py` **(20 pt.)**

점수의 총합은 100을 넘을 수 없습니다.

1 `utils.py` (30 pt.)

1.1 `is_numeral(s)`

주어진 문자열이 정수형 숫자가 될 수 있는지 판단하십시오. '-'로 시작하는 음수를 받을 수 있어야 합니다.

1.2 `is_varname(s)`

주어진 문자열이 변수의 이름이 될 수 있는지 판단하십시오. 변수의 이름은 소문자 알파벳 혹은 '_'로 이루어져 있어야 합니다.

1.3 `is_labelname(s)`

주어진 문자열이 라벨의 이름이 될 수 있는지 판단하십시오. 라벨의 이름은 대문자 알파벳 혹은 '_'로 이루어져 있어야 합니다.

1.4 `is_keyword(s)`

주어진 문자열이 이미 정의된 예약어(keyword)인지 판단하십시오. Plang의 예약어는 'print', 'input', 'jmp' 입니다.

1.5 is_getitem(s)

주어진 문자열이 list 참조(e.g., x[4])인지 판단하십시오.
TIP! 문자열에 '['가 적어도 한 개 존재하고 마지막 글자인 ']'가 첫번째 '[' 를 닫는 경우입니다.

1.6 in2post(operators, infix_tokens)

실습시간에 배운 차량기지(shunting yard) 알고리듬을 사용해 중위 표현식을 받아 후위 표현식을 반환하는 함수를 작성하십시오. 소괄호와 대괄호가 제대로 열리고 닫히는 것을 확인할 수 있어야 합니다. 만약, 소괄호가 제대로 여닫히지 않았다면 MismatchingParentheses 예외를, 대괄호가 제대로 여닫히지 않았다면 MismatchingBrackets 예외를 발생시켜야 합니다. 두 예외가 모두 발생할 수 있다고 판단되면 MismatchingParentheses를 발생시키십시오.
TIP! 소괄호('()')와 대괄호('[]')는 두 기호가 한 쌍이 되어 처리되어야 함을 주의하십시오.

2 petite_vm.py (class PTVM(VM)) (60 pt.)

virtual_machine.py에 위치한 class VM을 상속받는 class PTVM을 작성하십시오.

- 입력으로 주어지는 코드는 항상 띄어쓰기를 엄격히 지켜야 합니다.
- 한 줄에는 최대 하나의 명령만 있을 수 있습니다.
- 'jmp', 'input', 'print'는 변수의 이름이 될 수 없습니다.
- PTVM은 라벨을 제외한 모든 정수 값을 'INT_TYPE' 형태로 저장해야 합니다.
- 라벨의 값은 항상 'int'형이어야 합니다.
- 입력으로 주어지는 프로그램은 유한한 시간안에 멈춤을 가정합니다. (무한히 돌지 않습니다.)
- 더 자세한 내용은 4번 항목, 'Plang'을 참고하십시오.

2.1 scan_labels(self)

코드 전체에 위치한 라벨 정의를 읽고 그 위치를 기억합니다. 실제 코드의 첫 번째 줄은 1번 줄이지만, PTVM은 0번째 줄부터 세는 점을 잊지 마십시오.
NOTE! 라벨의 줄 위치는 'ixx' 타입이 아닌 파이썬 기본 'int'를 사용하십시오. 짧은 정수를 사용하면 코드 줄이 많은 경우 라벨 위치를 정확히 파악할 수 없습니다.

2.2 evaluate(self, expr)

표현식 expr의 값을 계산합니다. 이항연산자가 아닌 표현식들은 해당 함수에서 계산할 수 있어야 합니다. 이항연산자의 경우 이를 위한 메소드 compute(self, expr)를 사용합니다.

2.3 execute(self, pc, cmdline)

현재 VM의 PC와 명령줄을 입력 받은 후, 명령을 수행하고 다음 PC값을 돌려줍니다. 알 수 없는 명령의 경우 UnknownCommand 예외를 발생시킵니다.

2.4 run(self, breakpoints)

VM을 실행합니다. 먼저 모든 라벨들을 읽어 메모리에 저장합니다. 이후, VM의 PC가 마지막 줄 너머에 도달할 때 까지 명령을 수행합니다. 만약 실행 중 현재 PC가 set, breakpoints에 있는 숫자라면 현재 VM의 상태를 리스트에 추가하고 실행이 끝난 뒤 반환합니다. VM의 상태는 해당 위치에 있는 명령이 실행되기 전에 저장돼야 합니다.
NOTE! 현재 PTVM의 상태는 string 형태로 저장돼야 합니다.

TIP! 편집기에서 보이는 코드의 줄 번호와 PTVM이 인식하는 위치의 번호가 다름을 주의하십시오.
i.e., 첫 번째 줄 1→0(PTVM)

3 (Bonus) ixx.py (class ixx) (20pt.)

- 고정길이, 부호 있는(signed) 정수를 구현하는 ixx를 작성하십시오.
- 구현과정에서 int(·, 2)와 format(·, b)는 사용할 수 없습니다. 임의의 값을 10진 정수로 바꿔주는 int(·)는 사용 가능합니다.
- 연산자를 구현할 때에는, int의 동일한 연산자를 결과로 사용하실 수 없습니다.
(e.g., __mul__를 구현할 때에는 ixx(int(x) * int(y))를 반환할 수 없음. 계산 중간에 필요한 값을 계산하는 데에는 사용 가능)
- 구현을 완료한 후, PTVM의 INT_TYPE을 ixx 계열(e.g, i8, i32)로 바꿨을 때 동작이 되는지 확인하십시오.

3.1 __init__(self, value)

숫자의 2진 비트열을 튜플, self.__bitvec에 저장해야 합니다. value는 여러 타입의 값이 가능합니다.

1. int
10진수 정수 값을 받습니다. 이 값을 2진수로 바꾸어, self.__bitvec에 저장하십시오. 이 때, 작은 index에는 낮은 자리수의 값이 위치해야 합니다. 예를들어, self.__bitvec[0]에는 가장 작은 자리의 비트(lsb, least significant bit)가, 반대로 self.__bitvec[self.BIT_LENGTH-1]에는 가장 큰 자리의 비트(msb, most significant bit)가 위치해야 합니다. 만약 변환한 결과가 정해진 비트 수보다 적다면, 남은 부분은 0으로 채워야 합니다. 만약 주어진 값이 'ixx'의 표현 범위를 벗어난다면 OutOfCoverage 예외를 발생시키십시오.
TIP! 0으로 남은 부분을 채워주는 zext (zero extent) 가 이미 구현되어 있습니다.

2. list
2진 비트열이 들어오는 것을 가정합니다. 비트열의 모든 자리에 0 혹은 1이 있는지 확인하십시오. 0, 1 외의 값이 있는 경우 IllegalBitVector 예외를 발생시켜야 합니다. 정해진 자리수(self.BIT_LENGTH)보다 긴 리스트가 들어오는 경우에는, ~~작은쪽 끝에서부터 자리수만큼을 잘라 사용합니다.~~ 'OutOfCoverage' 예외를 발생시키십시오.

3. ixx의 파생 class
issubclass(type(value), ixx)를 통해 찾아낼 수 있습니다. 자신과 길이가 같거나 작은 정수형을 받을 수 있습니다. 이때, 부호를 유지하면서 빈 칸을 채워야 합니다.
TIP! 부호를 유지하면서 자리수를 늘리는 sext (signed extent) 가 이미 구현되어 있습니다.

4. 앞선 경우들이 아닌 경우, 마지막으로 int(value) 값으로 다시 생성(construct)을 시도합니다. 만약 실패한 경우에는 IllegalType 예외를 발생시켜야 합니다.

3.2 __int__(self)

비트열이 의미하는 숫자를 10진수로 반환하십시오. 이 메소드는 int(.)를 사용할 수 없습니다.

3.3 __neg__(self)

주어진 숫자의 부호를 반전하십시오. 2의 보수 표현 방식을 사용하므로, 일부 경우에 대해 Overflow 예외를 일으킬 수 있어야 합니다.

3.4 `__add__(self, other)`

- `other`은 `ixx`의 파생 class의 객체여야 합니다.
- `self`와 `other`의 길이가 다른 경우 큰 쪽을 따라야 합니다.
- 두 비트열을 각각의 비트 별로 더한 후, 더한 값이 1보다 큰 경우, 2로 나눈 몫을 그 위의 자리에 더해야 합니다. 그리고, 해당 위치에는 2로 나눈 나머지를 저장해야 합니다. 이렇게 아랫자리에서 윗자리로 전달되는 값을 carry라 합니다.
  마지막 자리(가장 큰 비트)에 전달되는 carry와 마지막 자리에서 나가야 하는 carry의 값이 서로 다른 경우에는 `Overflow` 예외를 발생시켜야 합니다.

3.5 `__sub__(self, other)`

`__add__`와 동일하게 `Overflow`를 발생시킬 수 있어야 합니다.

3.6 `__mul__(self, other)`

- `Overflow`를 발생시킬 수 있어야 합니다.
- 앞서 구현한 덧셈을 응용해 구현해야 합니다.
- 0에 두 수 중 '큰 수'를 '작은 수' 만큼 반복하며 더해야 합니다.

3.7 `__truediv__(self, other)`

- `DivideByZero`를 발생시킬 수 있어야 합니다.
- Python의 정수 나눗셈 '//'와 달리 소수점을 모두 버립니다.
  e.g., $-999/10 = -99.9 \rightarrow -99$, $10/-3 = -3.3... \rightarrow -3$
  TIP! 두 숫자의 부호 따로 절댓값 따로 계산하십시오.

3.8 `__abs__(self)`

숫자의 절댓값을 'ixx'계열로 반환해야 합니다.

3.9 `__bool__(self)`

가진 값이 0을 의미하면 `False`, 그 외의 경우에는 `True`를 반환해야 합니다.

3.10 `__{eq, ne, lt, gt, le, ge}__`

앞서 구현한 `__int__`와 `int`의 조건연산자들을 응용해 구현하십시오.

4 Plang

4.1 구성요소(Components)

Plang의 구성요소들은 다음과 같습니다.

1. x
   임의의 변수를 의미합니다. 변수의 값은 항상 정수 혹은 list여야 합니다. 변수 이름은 소문자 영어 알파벳과 '_'로만 구성될 수 있습니다.
   (e.g., '`snake_case`', '`rose_is_red`', '`the_rolling_stones`')

2. E
   임의의 표현식(expression)을 의미합니다. 표현식의 값은 항상 정수 혹은 list여야 합니다.
   (e.g., '`u != i`', '`23 + 5`', '`7 * x - 11`')

3. LABEL
   코드 상에서 현재 위치를 가리키는 이름입니다. 라벨 이름은 항상 대문자 영어 알파벳 혹은 '_'로 구성돼야 합니다. (e.g., 'SCREAMING_SNAKE_CASE:', 'PAINT_IT_BLACK:')

4.2 명령(Commands)

PTVM이 받아들이는 명령들은 다음과 같습니다.

1. `jmp E, LABEL`
   E의 값이 0이 아닐 때, '`LABEL`'의 위치로 이동합니다.
   (e.g., ''`jmp 1, EXIT`', '`jmp x != 4, HOME`', '`jmp m >= 121687, MASTER`')
   'E'의 값이 정수가 아닌 경우 '`Illegal Value`', 찾을 수 없는 라벨인 경우 `Unknown Label` 예외를 발생시켜야 합니다.

2. `LABEL:`
   현재 위치를 라벨로 지정합니다. 라벨은 `jmp`를 통해 도달할 수 있습니다. 라벨의 값(현재 위치 index)은 'INT_TYPE'이 아닌 'int'를 통해 저장해야 합니다.

3. `print(E)`
   'E'를 출력합니다. '`print`'와 '(' 사이에는 공백이 없음을 주의하십시오.
   (e.g., '`print(222)`', '`print(ECE)`', '`print(x + 4)`')

4. `x = E`
   변수 'x'에 'E'의 값을 저장합니다. 'E'의 값은 항상 정수여야 합니다. 아닌 경우에는 `IllegalValue` 예외를 발생시켜야 합니다.
   (e.g., '`x = 12`', '`y = x + 1`', '`z = x != y`')

5. `x = [E1; E2]`
   변수 'x'에 '`E2`'개의 연속된 '`E1`'을 갖는 `list`를 저장합니다. `E1, E2`의 값은 항상 정수여야 합니다. 만약 아니라면, `IllegalValue` 예외를 발생시켜야 합니다.
   (e.g., '`x = [0; 16]`', '`matrix_a = [1; 4 * 4]`')

6. `x[E1] = E2`
   list 변수 'x'의 'E1' 번째 index 위치에 정수, 'E2'를 저장합니다. 만약, 'x'가 list가 아니거나 'E2'의 값이 정수가 아닌 경우, `IllegalValue` 예외를 발생시켜야 합니다.
   (e.g., '`x[4] = 1 + 2`', `matrix_a[4 * i + j] = 0`)

- 한 줄에는 하나의 명령만 위치할 수 있습니다.
- 빈 명령줄이 가능함을 주의하십시오.
- '#'부터 해당 줄의 끝까지는 주석입니다.

4.3 표현식(Expression)

Plang의 표현식은 다음과 같습니다.

1. n
   임의의 10진 정수형 상수를 의미합니다. (e.g., 11, -7, 19721121)
2. x
   변수 'x'의 값을 의미합니다.
3. x[E]
   list 변수 'x'의 'E' 번째 값을 의미합니다. 'x'가 list가 아닌거나 'E'의 값이 정수가 아닌 경우 Illegal Value 예외를 발생시켜야 합니다.
4. (E)
   'E'와 동일한 값입니다. 연산 우선순위를 나타내기 위해 사용합니다.
5. E $\ominus$ E
   이항연산자 '$\ominus$'를 적용한 값입니다.
6. input()
   사용자로부터 키보드 입력을 받습니다. 입력은 항상 정수여야 합니다.

4.4 연산자(Operators)

이항연산자 $\ominus$는 다음과 같습니다.

1. '==' (__eq__): 같을 땐 1 다를 땐 0의 INT_TYPE
2. '!=' (__ne__): 다를 땐 1 같을 땐 0의 INT_TYPE
3. '<' (__lt__): 왼쪽이 더 작을 때 1의 INT_TYPE
4. '>' (__gt__): 왼쪽이 더 클 때 1의 INT_TYPE
5. '<=' (__le__): 왼쪽이 더 작거나 같을 때 1의 INT_TYPE
6. '>=' (__ge__): 왼쪽이 더 크거나 같을 때 1의 INT_TYPE
7. '+' (__add__): 둘 중 더 긴 타입의 INT_TYPE
8. '-' (__sub__): 둘 중 더 긴 타입의 INT_TYPE
   단항 연산자 '-'는 없음을 주의하십시오.
   (e.g., ' - expectation' (X) '0 - expectation' (O))
9. '*' (__mul__): 둘 중 더 긴 타입의 INT_TYPE
10. '/' (__truediv__): 정수 나눗셈의 몫. (ixx 사용 시, 왼쪽 타입의 ixx)
    Python의 정수 나눗셈과 달리 소숫점 부분을 모두 버리는 점에 유의하십시오.
    (e.g., -999 / 10 = -99 (C/C++, Plang) -999 / 10 = -100 (Python))

- 연산자와 피연산자(operand) 사이에는 항상 띄어쓰기가 있어야 합니다.
- 단항연산자 '-'가 없음을 주의하십시오. (e.g., '- x # 불가능', '0 - x # 가능')
- 나눗셈('/')은 항상 정수를 반환해야 합니다.

5 예외(Exceptions) - `exceptions.py`

예외 메시지의 대소문자 및 오타에 유의하십시오. 모든 예외는 해당 명령줄(command line)의 실행이 끝나기 전에 찾아낼 수 있어야 합니다. 또한, `NotImplementedError`를 제외한 모든 예외는 새로운 예외 class를 주어진 **상속** 계층에 맞게 정의하고 제공되는 기본 메시지를 설정하십시오.

- 아래 예외가 발생하는 경우 외에 문법적 오류는 없다고 가정합니다.
- `utils.py`의 경우는 'PTVM'과 별도로 채점하며 두 예외가 모두 발생할 수 있다 판단되면 소괄호를 우선으로 예외를 발생시키십시오. (e.g., (x[4]) $\Longrightarrow$ `MismatchingParentheses`)

5.1 `NotImplementedError`

구현되지 않았습니다. 해당 오류는 발생하면 안됩니다.

5.2 `PTVMException`의 sub-class들

1. `UnknownCommand` → `PTVMException`
   'Unknown Command': 알 수 없는 명령(command)입니다. Default 예외로 동작합니다.
2. `UnknownLabel` → `PTVMException`
   'Unknown Label': 정의되지 않은 라벨입니다.
3. `UnknownVariable` → `PTVMException`
   'Unknown Variable': 정의되지 않은 변수입니다.
4. `IllegalValue` → `PTVMException`
   'Illegal Value': 사용할 수 없는 값입니다. (e.g., 정수형 변수가 와야하는 위치에 list 변수가 오는 경우)
5. `SyntaxError` → `PTVMException` (`utils.py`)
   1) `MismatchingParentheses` → `SyntaxError`
      'Mismatching Parentheses': 소괄호가 제대로 닫히지 않았습니다.
   2) `MismatchingBrackets` → `SyntaxError`
      'Mismatching Brackets': 대괄호가 제대로 닫히지 않았습니다.
6. `NumericException` → `PTVMException` (`ixx.py`)
   1) `OutOfCoverage` → `NumericException`
      'Out of Coverage': `ixx`가 표현할 수 있는 범위의 값이 아닙니다.
   2) `IllegalBitVector` → `NumericException`
      'Illegal Bit Vector': 입력 비트열 중에 0과 1이 아닌 값이 있습니다.
   3) `IllegalType` → `NumericException`
      'Illegal Type': 연산에 사용된 변수들의 타입이 `ixx` 계열이 아닙니다.
   4) `Overflow` → `NumericException`
      'Overflow': 연산 결과가 타입이 표현할 수 있는 범위를 넘었습니다.
   5) `DivideByZero` → `NumericException`
      'Divide by Zero': `0` 으로 나눴습니다.

6 Appendix

6.1 Plang Example - `fibonacci.plang`

```
1  |     jmp 1, MAIN
2  |     # 1 1 2 3 5 8 13 21 34 55
3  |
4  | MAIN:
5  |     memo = [1; 10]
6  |     i = 2
7  | LOOP:
8  |     memo[i] = memo[i-1] + memo[i-2]
9  |     i = i + 1
10 |     jmp i < 10, LOOP
11 |
12 |     i = 0
13 | PRINT_LOOP:
14 |     print(memo[i])
15 |     i = i + 1
16 |     jmp i < 10, PRINT_LOOP
```

6.2 Abstract Syntax

$\ominus \longrightarrow + \mid - \mid * \mid / \mid \triangle$ Binary operators

$\triangle \longrightarrow == \mid != \mid < \mid > \mid <= \mid >=$ Relational operators

$E \longrightarrow n$
$\quad \mid x$
$\quad \mid x[E]$
$\quad \mid (E)$
$\quad \mid E \ominus E$
$\quad \mid \text{input}()$

$C \longrightarrow C$ `'newline'` C	Sequence
$\quad \mid l:$	Label definition
$\quad \mid x = E$	Assignment
$\quad \mid x[E] = E$	Assignment (list element)
$\quad \mid x = [E; E]$	Integer list definition
$\quad \mid \text{jmp } E, l$	Conditional jump
$\quad \mid \text{print}(E)$	

$P \longrightarrow C$ Program