Esercitazione 10

19 dicembre 2023

Lo scopo di questa esercitazione è quello di realizzare un sistema per costruire e valutare espressioni. Il sistema dovrà sfruttare il fatto che è possibile definire gerarchie di classi per evitare la duplicazione di codice.

L'idea principale è quella di realizzare un sistema tale per cui una espressione come la seguente:

```
2 x + 3 *
```

possa venire interpretata come $(2 + x) \times 3$. Per fare questo è necessario utilizzare uno stack (la cui implementazione è fornita nella classe `Stack`) in cui:

- Se si incontra una stringa interpretabile come un intero si crea un oggetto di tipo `Constante` si inserisce nello stack;
- Se si incontra una stringa non corrispondente ad alcuna funzione (e.g., `x` o `y`), si crea un oggetto di tipo `Variable` e si inserisce nello stack;
- Se si incontra una stringa corrispondente a un operatore `op` (e.g., `+`), si verifica l'arità k della funzione (e.g., `2` per `+`), si effettuano k pop dallo stack per ottenere gli argomenti di `op`, si costruisce un oggetto di tipo `op` (e.g., `Addition`) e si inserisce nello stack

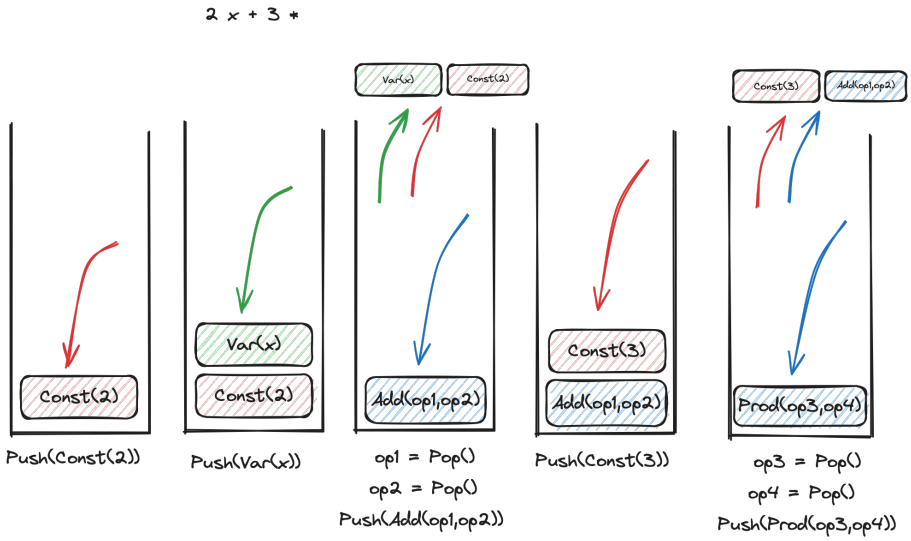

Figure 1: Costruzione di una espressione a partire dalla stringa `2 x + 3 *`

Se la stringa rappresentante l'espressione è corretta allora alla fine della lettura della stringa di input lo stack conterrà un solo elemento corrispondente all'espressione rappresentata dalla stringa.

Implementazione

È necessario implementare le seguenti classi:

- `Expression`. Rappresenta una espressione generica. Deve avere un metodo di classe `from_program(cls, text, dispatch)` che, dato un testo rappresentante espressione in notazione polacca inversa ritorna un oggetto di tipo (una sottoclasse di) espressione.
- `Variable`. Rappresenta una variabile. Il nome è indicato quando viene costruito l'oggetto. Se durante la valutazione l'ambiente non ha un valore per la variabile allora deve essere sollevata l'eccezione `MissingVariableException`.
- `Constant`. Rappresenta una costante il cui valore è fornito al momento della costruzione.
- `Operation`. Rappresenta una operazione astratta. Il costruttore riceve gli argomenti dell'espressione sotto forma di una lista.
- `BinaryOp` e `UnaryOp` specializzano `Operation` rispettivamente ai casi binario e unario
- Le seguenti operazioni binarie (e la corrispondente forma testuale):

`Addition`	`+`	$x + t$
`Subtraction`	`-`	$x - y$
`Multiplication`	`*`	$x \times y$
`Division`	`/`	x/y
`Modulus`	`%`	$x \bmod y$
`Power`	`**`	x^y

- Le seguenti operazioni unarie (e corrispondente forma testuale):

`Reciprocal`	`1/`	$1/x$
`AbsoluteValue`	`abs`	$\lvert x \rvert$

I metodi da implementare sono i seguenti:

- `evaluate(self, env)`: valuta l'espressione usando gli assegnamenti di variabili nel dizionario `env`, che ha associazioni tra nomi di variabili (stringhe) e valori;
- `__str__(self)` per la rappresentazione sotto forma di stringa;
- `op(self, x, y)` (per operazioni binarie) e `op(self, x)` per operazioni unarie. Questi metodi implementano le operazioni definite dalla loro classe (e.g., la somma per `Addition`). Questi metodi hanno senso di esistere solo per le classi che rappresentano operazioni binarie;
- Ogni classe (non singola istanza) specializzata deve avere un variabile `arity` per specificare l'arità della funzione.

La scelta di dove implementare i singoli metodi è lasciata libera.

Note

- Nella conversione da stringa a espressione l'argomento `dispatch` è un dizionario che associa a delle stringhe la corrispondente classe da utilizzare. Per esempio a `+` assocerà `Addition`. In questo modo diventa possibile costruire facilmente l'espressione corrispondente: quando si incontra `+` si vede che la classe corrispondente ha attributo `arity` pari a `2`, quindi si effettua la rimozione di `2` elementi dallo stack, si crea l'oggetto rappresentante l'addizione passando questi due argomenti al costruttore
- Il metodo `split()` disponibile per le stringhe ritorna un array di stringhe ottenute spezzando la stringa di partenza sui separatori (di default lo spazio). Quindi `"a b c".split()` ritorna `['a', 'b', 'c']`. Questo può essere utile a gestire la stringa di input che deve essere trasformata in una espressione.
- Notate che è possibile forzare una sottoclasse (se deve funzionare correttamente) a implementare uno dei metodi della classe base facendo sollevare alla chiamata del metodo l'eccezione `NotImplementedError`.