29 października 2024

Kurs rozszerzony języka Python

Lista 4.

Proszę wybrać jedno z zadań, a jeśli ktoś wybrał zadanie 1, należy wybrać jeden z wariantów (A), (B) lub (C); wystarczy zrobić jeden wariant. Każde zadanie jest warte 6 punktów.

Implementację uzupełnij o własne wyjątki (klasy) reagujące na niepoprawne sytuacje, np. brak przypisania wartości zmiennej.

Zadanie 1.

Zaprogramuj klasę ***Wyrazenie*** wraz z odpowiednimi podklasami reprezentującymi różne rodzaje wyrażeń arytmetycznych. Przykładowo, wyrażenie

```
(x + 2) * y
```

może być reprezentowane jako

```
Razy(Dodaj(Zmienna("x"), Stala(2)), Zmienna("y"))
```

gdzie ***Razy***, ***Zmienna*** czy ***Stala*** są odpowiednimi podklasami klasy ***Wyrazenie***.

Zaprogramuj w każdej klasie metody

- `oblicz(self, zmienne)`, która oblicza wartość wyrażenia; przy czym argument `zmienne` jest słownikiem, gdzie kluczami są nazwy zmiennych, a wartościami wartości tych zmiennych;
- `__str__` zwracającą jako string ładnie sformatowane wyrażenie;
- `__add__` i `__mul__`, które dla dwóch obiektów $w1$ i $w2$ klasy ***Wyrazenie*** tworzy nowy obiekt ***Wyrazenie***, który reprezentuje odpowiednio sumę bądź iloczyn $w1$ i $w2$.

Zaprogramuj reakcję na niepoprawne dane poprzez zgłoszenie odpowiednich wyjątków. Dla każdego rodzaju błędu zaprogramuj własną klasę wyjątków np. `VariableNotFoundException`.

Wymagane jest, aby były zdefiniowane stałe, zmienne i podstawowe działania arytmetyczne.

Implementację powyższego zadania rozszerz na jeden z poniższych sposobów.

(A) Można stworzyć podobną hierarchię klas reprezentującą prosty język programowania z przynajmniej instrukcjami: instrukcją przypisania, instrukcją warunkową `if` i instrukcją pętli `while`. Przydatna będzie też instrukcja reprezentująca ciąg instrukcji. Można przyjąć, że wyrażenie arytmetyczne równe zero interpretujemy jako fałsz, a prawdę w przeciwnym przypadku. W każdej z tych klas trzeba zaprogramować metodę `wykonaj(self, zmienne)` wykonującą kolejne instrukcje.

Przyda się też metoda `__str__` zwracająca jako string czytelnie sformatowany program.

(B) Mając wyrażenia w postaci takiego drzewka, można pokusić się o uproszczenie wyrażenia, przykładowo wyrażenia zawierające tylko stałe typu `2 + 2 + "x"` można uprościć do `4 + "x"`, albo skorzystać z własności mnożenia przez zero. Zaprogramuj przynajmniej dwie takie reguły upraszczające w formie metody klasy klasy ***Wyrazenie***. Operacja uproszczenia ma utworzyć nowy obiekt klasy ***Wyrażenie***.

(C) Wyrażenia zawierające tylko jedną zmienną możemy traktować jako funkcje. Zaprogramuj metodę klasy wyliczającą wyrażenie (obiekt klasy ***Wyrazenie***), będące pochodną funkcji.

Zadanie 2.

Zaprogramuj klasę ***Formula*** wraz odpowiednimi podklasami, które będą reprezentować formuły zdaniowe. Przykładowo

$$\neg x \vee (y \wedge \texttt{true})$$

może być przedstawione jako

```
Or(Not(Zmienna("x")), And(Zmienna("y"), Stala(True)))
```

Przyjmujemy, że formuły składają się ze stałych `True` i `False`, zmiennych oraz przynajmniej alternatywy, koniunkcji i negacji.

Zaprogramuj w każdej klasie metody

- `oblicz(self, zmienne)`, która oblicza wartość wyrażenia; przy czym argument `zmienne` jest słownikiem, gdzie kluczami są nazwy zmiennych, a wartościami wartości tych zmiennych, `__str__` zwracającą jako string sformatowane wyrażenie w postaci infiksowej;
- `__add__` i `__mul__`, które dla dwóch obiektów $w1$ i $w2$ klasy ***Formula*** tworzy nowy obiekt ***Formula***, który reprezentuje odpowiednio alternatywę bądź koniunkcję $w1$ i $w2$.
- `.tautologia()` która sprawdza, czy podana formuła jest tautologią. Można wykorzystać tu pakiet `itertools` poznany na poprzednich zajęciach.

Zaprogramuj metodę (może to być np. metoda klasy), która dla danej formuły wylicza jej uproszczenie korzystając z zależności

$$p \wedge \mathit{false} \equiv \mathit{false}$$

czy

$$\mathit{false} \vee p \equiv p$$

Marcin Młotkowski