PyGDB — analiza strukture izvornog koda

Projekt iz kolegija Napredne baze podataka

Luka Šimek

Zagreb, 14. svibnja 2024.*

Sadržaj

1 Uvod **1**
1.1 Motivacija 1
1.2 Python i njegove značajke 1

2 Alati **2**
2.1 Apstraktna sintaksna stabla 2
2.2 Tablice simbola 3

*datum zadnje izmjene: 25. svibnja 2024.

1 Uvod

1.1 Motivacija

Suvremeni softverski paketi često u sebi sadrže nepregledno veliku količinu raznih elemenata, od modula, klasa i funkcija do samog broja linija koda. Kao korisnicima, često nam se korisno malo bolje upoznati s arhitekturom paketa s kojim radimo, bilo da je zbog što kvalitetnijeg korištenja njegovih funkcionalnosti ili debuggiranja vlastitog koda.

S tim ciljem, pogodno bi bilo imati način za analizirati strukturu nekog paketa vizualno i potencijalno iz ptičje perspektive. Također, htjeli bismo da ta analiza bude fleksibilna — kao što smo naveli, cjelokupni sadržaj paketa može biti vrlo nepregledan, stoga želimo istovremeno biti u mogućnosti vizualizirati različite module u paketu, ali i podatke o tek nekolicini konkretnih funkcija.

Primijetimo da se spomenuta struktura prirodno preslikava na graf. Za vrhove grafa uzimamo različite objekte (potpakete, module, klase, funkcije i ostale varijable), dok su bridovi dani njihovim raznim odnosima (klasa je definirana u modulu, funkcija je metoda klase, jedna klasa naslijeđuje od druge itd.)

Naši zahtjevi — sakupljanje velike količine podataka o paketu ili više njih, fleksibilno dohvaćanje raznih dijelova tih podataka i modeliranje preko grafa navode nas na korištenje grafovske baze podataka. Konkretno, u ovom projektu koristit ćemo Neo4j. Jedan od razloga za to je da je riječ o trenutno najpoznatijem sustavu za upravljanje grafovskim bazama podataka. Cypher, njegov jezik za upite, sličan je poznatom SQL-u u sinktaksi, ali i sam po sebi vrlo kvalitetan i pogodan za upite nad grafovskim bazama. Dodatno, Neo4j Browser u sebi već ima ugrađeno korisničko sučelje i mogućnosti vizualizacije rezultata upita.

1.2 Python i njegove značajke

Iako različiti programski jezici mogu dijeliti implicitnu shemu takve grafovske baze, algoritam za transformaciju programskog koda u graf nužno će ovisiti o programskom jeziku u kojemu je kod pisan. Ni zajednička shema nije nužna. Primjerice, već smo nekoliko puta spomenuli klase, no klase nisu dio svih programskih jezika.

U ovom projektu koristit ćemo Python i baviti se paketima pisanima, barem prvenstveno, u Pythonu. Jedan razlog svakako je u popularnosti jezika — Python je popularan jezik općenite uporabe i koristi se u velikom broju otvorenih paketa. Posebno spomenimo cijeli *data science* ekosustav u kojemu je Python zauzeo jednu od vodećih uloga u izradi biblioteka za znanstveno računanje, obradu i vizualizaciju podataka, statistiku i statističko učenje.

Činjenica da je Python dinamički tipiziran predstavlja izazov, ali i povećava smisao ovakvog projekta. U Pythonu ne postoje varijable u tipičnom smislu, već postoje samo *imena* koja možemo pridružiti objektima. To pridruživanje je potpuno fleksibilno. Primjerice, sljedeći kod je sasvim pravilan:

```
class A:
  pass

a = A()
A = 3
```

Dakle, dinamički je određeno na koji se objekt odnosti oznaka `A`. U ovom projektu zadržavamo se na statičkoj analizi koda — dinamička analiza bila bi skuplja vremenski i memorijski te bi bila dosta kompleksnija. Dodatno, bila bi ovisna o ispravno postavljenom i funkcionalnom softverskom okruženju. Zauzvrat, naši rezultati ne moraju uvijek biti pouzdani, ali se situacija kao iznad gotovo nikad neće naći u praksi, među poznatim i kvalitetno napravljenim paketima.

Treći razlog za Python je njegova opsežna standardna biblioteka koja će nam omogućiti interakciju sa strukturama podataka koje inače koristi samo kompajler. Više o tome ćemo reći u sljedećem poglavlju.

2 Alati

2.1 Apstraktna sintaksna stabla

Apstraktno sintaksno stablo (eng. *abstract syntax tree*, *AST*) je struktura podataka koja predstavlja strukturu programskog koda u nekom programskom jeziku. Vrhovi tog stabla odgovaraju sintaktičkim konstruktima u jeziku. Primjerice, vrh može odgovarati binarnoj operaciji. U tom slučaju lijevo i desno podstablo odgovaraju operandima, koji, rekurzivno, sami mogu imati više binarnih operacija u sebi. Vrh može odgovarati for-petlji — tada su sva djeca podstabla koja odgovaraju konstruktima unutar bloka određenog for-petljom. Ostali važni podaci, kao što su varijabla i granice iteracije, mogu biti spremljeni kao atributi tog objekta ili ponovo kao djeca glavnog vrha.

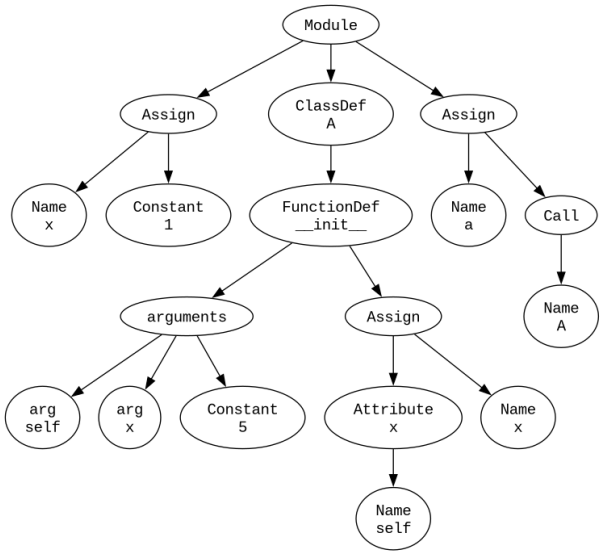
Apstraktna sintaksna stabla su međukorak prilikom kompilacije programskog koda u većini modernih programskih jezika. Tako i Pythonov kompajler stvara AST prije eventualnog prijevoda u *bytecode*. Apstraktna sintaksna stabla su dio samog kompajlera, ali modul `ast.py` (službena dokumentacija pod [2], podrobnija dokumentacija pod [3]) u standardnoj biblioteci služi kao omotač koji nudi AST kao objekt u Pythonu s nekoliko pomoćnih funkcija za njihovu obradu.

Postojeće funkcionalnosti modula `ast.py` dozvoljavaju pregled stabla u formatu sličnom JSON-u (`ast.dump`), ali pomoću paketa Graphviz ih možemo vizualizirati grafički. Razmotrimo sljedeći isječak koda:

```
x = 1
class A:
    def __init__(self, x=5):
        self.x = x

a = A()
```

Odgovarajuće apstraktno sintaksno stablo dano je na slici 2.1. Tekst u čvorovima odgovara različitim klasama u modulu, potklasama bazne klase `ast.AST` (npr. `ast.ClassDef` odgovara definiranju klase), a dodatne informacije, kao što je ime objekta ili vrijednost konstante, dane su ispod. Najčešće, čvorovi imaju još relevantnih atributa, ali na slici nisu prikazani u svrhu preglednosti.



Slika 1: Primjer AST-a

2.2 Tablice simbola

Tablica simbola (eng. *symbol table*) je još jedna struktura podataka koju koriste kompajleri. Generira se nakon AST-a, te u Pythonovom slučaju direktno prethodi konačnoj kompilaciji. Kao i AST, moguć joj je pristup kao objektu u Pythonu preko modula `symtable.py` (službena dokumentacija pod [4]) iz standardne biblioteke.

Tablica simbola sadrži sve simbole u kodu, a posebno se pritom osvrće na njihove opsege (*scope*-ove) i eventualno neke dodatne atribute. Pritom, glavna tablica sadrži globalni nazivni prostor (*namespace*), a može sadržavati i podtablice sa vlastitim, lokalnim nazivnim prostorima. U Pythonu, to je slučaj za nazivne prostore klasa i funkcija, ali ne, za razliku od drugih jezika, for-petlji ili sličnih blokova koje nemaju vlastiti nazivni prostor.

Budući da se tablica generira iz samog AST-a, sve što možemo napraviti s tablicama simbola možemo i sa samim AST-om. Ipak, uporaba modula iz standardne biblioteke smanjuje mogućnost greške i osigurava usklađenost sa samim Pythonovim kompajlerom. Za primjer toga što je sve dostupno u Pythonu, čitatelj može pogledati primjer ispisa u bilježnici

`example/symtable_example.ipynb`

u repozitoriju projekta [1].

Literatura

[1] Luka Šimek (lsimek2). *nbp-projekt*. 2024. URL: https://github.com/lsimek2/nbp-projekt.

[2] *ast – Abstract Syntax Trees*. Python Software Foundation. URL: https://docs.python.org/3/library/ast.html (pogledano 14. 5. 2024.).

[3] Thomas Kluyver. *Green Tree Snakes - the missing Python AST docs*. 2012. URL: https://greentreesnakes.readthedocs.io (pogledano 19. 5. 2024.).

[4] *symtable — Access to the compiler's symbol tables*. Python Software Foundation. URL: https://docs.python.org/3/library/symtable.html (pogledano 25. 5. 2024.).