Drugi kolokvij

Interpretacija programa

10. veljače 2017.

Ovo je "*open book*" kolokvij. Dozvoljeno je korištenje bilo kakvih materijala — bilješke s vježbi, Python help, tutoriali, postovi na online forumima,... — **nastalih prije** kolokvija (npr. dozvoljeno je na *StackOverflow*u naći rješenje nekog zadatka, ali nije dozvoljeno tamo postaviti pitanje kako se rješava neki zadatak). Također, nije dozvoljena komunikacija (razgovor, chat, razmjena bilježaka) **među** studentima.

Potrebne / korisne "hint-datoteke" nalaze se u Github repozitoriju, `http://github.com/vedgar/ip` — prethodno skinite repozitorij! Rješenja zadataka pišite **isključivo** u datoteke `Z1.py` i `Z2.py` (početne verzije tih datoteka već se nalaze u repozitoriju), te na kraju obje datoteke *upload*ajte na `http://ted.math.hr`. Odgovore na pitanja koja se nalaze u zadacima pišite u komentare unutar koda.

Svaka datoteka, uključivo sa svim testovima koji su inicijalno u njoj, treba se moći izvršiti u Pythonu bez grešaka. [Smijete pisati i u nekom drugom programskom jeziku, ali u tom slučaju pišete "od nule". Možete koristiti i neke alternativne reprezentacije objekata iz repozitorija (konačni automati, regularni izrazi,...) ako već postoje na internetu. U tom slučaju citirajte link s kojeg ste to skinuli.] Ako imate neki kod koji po Vašem mišljenju pokazuje ideju rješenja, ali iz nekog razloga ne radi, napišite ga u komentar. Korisno je u datoteku uključiti još neke testove pored onih obaveznih.

Maksimalno vrijeme rješavanja je 180 minuta. Prvi zadatak vrijedi 15 bodova + 5 bonus bodova, drugi 20 bodova. Sretno!

Veky

Prvi zadatak

- [3b] Implementirajte funkciju `k_komplement`, koja prima konačni automat *M* nad abecedom *Σ* i vraća konačni automat koji prihvaća jezik *Σ* \ L(M)*.
- [4b] Implementirajte funkciju `k_simetrična_razlika` koja prima konačne automate *M1* i *M2* i vraća konačni automat koji prihvaća jezik *L(M1)△L(M2)*.
- [2b bonus] Funkcija iz prethodne točke može se implementirati koristeći samo funkcije Kartezijeve konstrukcije za presjek i uniju (koje su već napravljene u datoteci `KA.py`), te funkciju `k_komplement`. Ako *M1* ima n_1 stanja, a *M2* ima n_2 stanja, koliko stanja ima tako dobiveni automat za simetričnu razliku?
- [3b bonus] Možete li optimizirati funkciju `k_simetrična_razlika` tako da rezultatni konačni automat ima samo $n_1 \cdot n_2$ stanja? Objasnite zašto to nije moguće, ili napišite takvu funkciju.
- [5b] Implementirajte funkciju `prazan_jezik_KA` koja rješava problem praznosti jezika za konačne automate: prima konačni automat *M* i vraća je li *L(M)*=∅. Neobavezna uputa: funkcija `optimizirana_partitivna_konstrukcija` (iz `NKA.py`) uklanja sva nedostupna stanja zadanog NKA.
- [3b] Implementirajte funkciju `jednaki_jezici_KA` koja rješava problem jednakosti jezika za konačne automate.

Drugi zadatak

- [2b] *Specijalni* znak je naziv za otvorenu ili zatvorenu zagradu, vertikalnu ili kosu crtu te zvjezdicu. Napišite funkciju `specijalan` koja prima znak i vraća je li specijalan. (Kosa crta je `/`, a ne `\`. Zagrade su samo oble `()`. Primijetite: razmak nije specijalan).
- [2b] Svakom specijalnom znaku odgovara zasebni tip tokena. Svim ostalim znakovima odgovara jedan tip tokena `ZNAK`, čiji sadržaj kaže o kojem se znaku radi. Svi tokeni (osim `KRAJ`) imaju sadržaj duljine 1. Napišite odgovarajuće članove `Enum`-klase `RI`.
- [4b] Napišite funkciju (generator) `RI_lex` koja `yield`a tokene čiji tipovi su gore navedeni, a čiji sadržaji predstavljaju znakove ulaznog stringa.
- [4b] *Regularni izrazi* grade se od elementarnih jezika koji su predstavljeni tokenom `ZNAK`, te praznog jezika `/`. Grade se (rastućim prioritetom, kojeg možemo mijenjati zagradama) unijom (binarni infiksni operator `|`), konkatenacijom, te zvjezdicom (unarni postfiksni operator `*`). Napišite odgovarajuću beskontekstnu gramatiku nad abecedom koju čine tipovi tokena. [Za -1b: smijete pretpostaviti da se `ZVIJEZDA` ne pojavljuje više puta uzastopno (da riječ *r*`**` nije u jeziku).]
- [6b] Svaki nespecijalni znak *α* predstavlja jezik koji sadrži samo jednoslovnu riječ *α*. Izraz `(`*r*`)` predstavlja isti jezik kao izraz *r*, dok izraz *r*`*` predstavlja Kleenejevu zvijezdu tog jezika. Napišite klasu `RIParser` tako da radi funkcija `RI_parse` (već napisana u `Z2.py`) koja prima string i vraća `RegularanIzraz` kojeg on predstavlja. [Za -1b: funkcija smije vratiti `tip.AST` strukturu umjesto objekta klase `RI.RegularanIzraz`.]
- [2b] Prijavite korisne poruke o greškama (navodeći poziciju greške, koji tokeni su očekivani, a koji je pročitan) ako argument od `RI_parse` ne predstavlja validan

regularan izraz. (Ovo će biti trivijalno ispunjeno ako koristite `Buffer` odnosno `Parser` iz `tip.py`.)