MR. ROBOT (PhD)

MR. ROBOT predstavlja višenamjensko programsko okruženje za upravljanje robotom u zadanom dvodimenzionalnom skladištu na interaktivan način. Robot se po skladištu može kretati, mijenjati stanje skladišta podizanjem i spuštanjem kutija i popravkama električnih kvarova te analizirati svoju okolinu putem senzora za objekte na neposrednoj i većoj udaljenosti te senzora za intenzitet zvuka i vlastite lokacije.

Tri test primjera koja su izvedena, izvršena i analizirana odgovaraju trima situacijama koje je naručitelj projekta, tvrtka **fsociety** zatražila.

Prvi test primjer odgovara opasnoj situaciji u kojoj su neprijateljske snage uspjele pristupiti kamerama u skladištu u kojem se nalaze kutije sa svom (papirnatom i električnom) dokumentacijom napada na sustave. Ono što neprijatelji ne znaju je da je tvrtka podmetnula kutije praznog sadržaja i skladište slabog sustava zaštite da ih skrene s pravog traga i pribavi sebi dragocjeno vrijeme potrebno za zamesti tragove svojih drugih aktivnosti. Nakon što su to uspješno odradili, žele poslati poruku svojim neprijateljima koji to skladište mogu vidjeti i u skladu s time im, nakon što su otklonili sve električne kvarove u skladištu da ne riskiraju požar i stradanje nevinih, kutijama ispisuju poruku *IP*.

Za drugi test primjer, tvrtka se ipak našla u ozbiljnijoj situaciji. Nakon podrugljive podvale sa ispisom slova kutijama, neprijateljske snage presrele su signal koji je tvrtka slala tijekom komunikacije s robotom te su uspjeli podmetnuti nekoliko očitih kvarova koji su električnog tipa kao i u prethodnom primjeru, ali isto tako i moderne naprave koje svojom jačinom zvuka robote mogu uništiti ii oštetiti. Njihov je zvuk na frekvenciji koju ljudsko uho ne čuje, a isto tako izvori zvuka su nevidljivi, odnosno tvrtka ih ne može opaziti preko kamere, a robot će biti uništen ako im dođe preblizu. Konkretno, eksperimenti obavljeni u mirnodopskom razdoblju pokazali su da slobodni robot može podnijeti do 70 dB, a kada nosi kutiju do 50 dB štetnog zvuka. Cilj ovog test primjera je pomicanje svih kutija na sigurni dio skladišta za dano skladište tako da bi se na siguran način mogle prenijeti u drugo skladište, ali imajući na umu to da nam je od interesa sačuvati zdravlje robota.

Konačni test primjer zatražen je nastavno na ekstrakciju svih kutija iz skladišta u koje su neprijatelji prodrli u neko od dostupnih novih. Kako su kamere očigledno podložne sigurnosnim poteškoćama, naručitelj je autore jezika zatražio da pripreme program u kojem se robot pomiče po zadanom skladištu te na spiralan način provjerava postoje li blokovi određene veličine tako da u njega mogu prenijeti i veće kutije. Tvrtki je nužno bilo samo da se test primjer implementira za dano skladište te će ga, kao sposobni programeri, prilagoditi općoj namjeni jednom kada imaju potrebnu funkciju za spiralno pretraživanje.

Struktura problema

Robot je prikazan na dvodimenzionalnoj mapi skladišta kao jedna od strelica (ovisno o smjeru '↑', '←', '↓' ili '→') ako je slobodan, odnosno jedna od usmjerenih punih strelica (ovisno o smjeru '▲', '◀', '▼' ili '▶') ako nosi kutiju. Prazna mjesta na mapi predstavljena su znakom '*' te se robot može kretati isključivo po njima i na njih može stavljati dignute kutije. Isto tako, električni kvarovi predstavljeni znakom '₸' nestaju jednom kada ih robot popravi, a štetni izvori zvuka označeni znakom '♫' nisu vidljivi robotu te ih jedino može opaziti putem senzora za intenzitet zvuka i ne smije im se previše približiti.

Kostur programa i podržani tipovi

Programi danog jezika započinju deklaracijama nula ili više funkcija, od kojih posljednja mora biti `main` i može biti jednog od korisnički definiranih povratnih tipova `Number`, `Bool` i `List` koji se vraćaju putem naredbe `return`. Deklaracije svih funkcija započinju ključnom riječi `def` te se njihova tijela nalaze unutar standardnog C-ovskog bloka između vitičastih zagrada. U sklopu jezika podržani su i rekurzivni pozivi funkcija svih povratnih tipova, uz izuzetak funkcije `main` te pozivanja jedne rekurzije unutar druge. Nizovi naredbi koje sačinjavaju blokove također se pozivaju u skladu s paradigmom jezika C, odnosno redom ime funkcije ili naredbe, praćeno s nula ili više argumenata unutar oblih zagrada te konačno i točka-zarezom. Prilikom parsiranja koda, osigurava se i konzistentnost tipova koji se pridružuju, šalju i vraćaju, ali isto tako je predviđena i omogućena konverzija svih dostupnih tipova u sve druge. Na tipovima su također implementirane i standardne aritmetičke operacije i relacije navedene u zahtjevima naručitelja projekta, no uz njih su standardne logičke konstante i operacije proširene na Kleenejevu trovaljanu logiku sa konstantama `True`, `False` i `Unknown` te logičkim veznicima ¬ (`not`), ∧ (`and`) i ∨ (`or`). Uz proširenja logičkih, za lisne tipove su, osim inicijalizacije, pretvorbe u druge tipove i pridruživanja, implementirane mogućnosti dodavanja na kraj liste (`append`), na njen početak (`add`) te pristupanje njenom proizvoljnom elementu putem metode `get`. Valja napomenuti i da su pridruživanja proširena i na složenije izraze, kao što se da zaključiti iz gramatike jezika, no da se ova funkcionalnost nije proširila na liste jer nije potrebna tijekom izvršavanja primjera, no da se može implementirati korištenjem dodatne memorije u kojoj bi se rekurzivno spremali tipovi ugniježđenih listi. Naravno, naglašavamo da pristupanje elementima i sve zahtijevane funkcionalnosti (a i dodatne koje su navedene) poput ugniježđenih listi rade besprijekorno te da se upravo jedna takva lista i vraća u posljednjem test primjeru.

Grananje, petlje i nelokalna kontrola toka

Jezik podržava standardno `if` grananje sa oznakama kao u npr. Pythonu (`if`, `elif` i `else`) sa blokovskom strukturom i oblim zagradama za logičke uvjete kao u C-u. S druge strane, petlje `for` i `while` i oblikom i rječnikom odgovaraju C-ovskoj paradigmi uz trivijalno ograničenje da se inicijalizacija varijable vrši prije ulaska u petlju koje, naravno, ne narušava ekspresivnu moć jezika. Nelokalna kontrola toka implementirana je po uzoru na veprovsku (`vepar.py`) pomoću naredbe `break` (a i spomenute `return`) te provjereno prekida izvođenje isključivo unutar petlji (izvan njih vodi na pogrešku kao što i treba).

Korisničke instrukcije i novoimplementirane naredbe

Uz očekivane naredbe poput inicijalizacije i sl., korisniku je dostupno i više naredbi specifičnih jeziku.

Osnovna funkcionalna mogućnost robota je njegovo kretanje po dvodimenzionalnoj mapi zadanog skladišta koje se vrši naredbom `forward` koja prima cjelobrojni parametar koji predstavlja udaljenost za koju želimo da se robot pomakne. Robot se može kretati samo po slobodnim mjestima označenim znakom '*' i nikojim drugima, a kao povratnu vrijednost funkcije `forward` daje ukupan broj mjesta za koji se uspio pomaknuti.

Uz ovo, kao korištenje podržane trovaljane logike, naredbom `check` poziva se robotov interni senzor koji na zadanoj udaljenosti ispred njega (jer funkcionira na temelju odašiljanja elektromagnetskih valova, a robot detektor za njih ima samo na svojoj prednjoj strani) može provjeriti jesu li pozicije pred njim slobodne (ovisno o tome vraća `True` ili `False`), a na udaljenosti većoj od te kao povratnu logičku vrijednost daje `Unknown` jer to nije unutar dometa njegovog senzora.

Za promjenu smjera kretanja robota koristimo naredbu `turn` koja ga može okrenuti za cjelobrojni višekrantik kuta π koji prima kao parametar te kao povratnu vrijednost u našoj inačici uvijek vraća `True` jer ne postoji mogućnost da se robot ne može okrenuti. Mogućnost povrata različitih logičkih vrijednosti ostavljena je za buduća proširenja jezika i mogućnosti robota.

Kao prvu vrstu interakcije, robot kutije u skladištu može podizati (naredba `pick` bez argumenata), kretati se s njima te ih spuštati (naredba `put` bez argumenata) na slobodna mjesta. Važno je reći da robot kutiju može podignuti ako i samo ako se nalazi neposredno blizu nje te je usmjeren prema njoj. Isto tako, jednom kada podigne kutiju, ne može podignuti drugu ni popravljati kvarove (vidi idući paragraf) dok je ne spusti. Također, dok drži kutiju, njegova slika na mapi promijeni se iz usmjerene strelice u usmjereni ispunjeni trokut.

U skladištu također može doći do pojave električnih kvarova koje robot može popraviti (naredba `repair`), nakon čega oni bivaju uklonjeni s mape. S obzirom da robot također funkcionira na principu električne energije, kvarovi također predstavljaju lokacije po kojima se ne može kretati. Kao i za podizanje kutija, da bi robot popravio kvar, mora se nalaziti neposredno blizu njega te biti usmjeren ka njemu.

Robot je sposoban analizirati svoju okolinu, pa su tako uz spomenutu naredbu `check` također implementirane funkcije `iswall`, `isbox` i `isshort` koje vraćaju logičku vrijednost ovisno o tome je li ono što se nalazi neposredno ispred robota (i čemu je usmjeren) redom zid, kutija ili električni kvar.

Ponovno, uz naredbu `check` čiji je domet veći od jednog razmaka na mapi, robot ima i mogućnost analize intenziteta zvuka štetnih naprava u svojoj okolini putem naredbe `noise` koja kao povratnu vrijednost ima cijeli broj koji odgovara intenzitetu najsnažnijeg štetnog izvora zvuka u okolini robota na zadanoj maksimalnoj udaljenosti.

Konačno, robot putem globalnog pozicionirajućeg sustava i naredbi `getx` i `gety` može dohvatiti i upravljaču (korisniku) proslijediti svoje trenutne relativne koordinate unutar skladišta.

Miscellaneous

U svrhu prirodnijeg načina opisa koda, autori jezika implementirali su jednolinijske komentare koji započinju znakom '#' i protežu se do kraja reda te višelinijske koji započinju stringom "/*" i završavaju sličnim stringom "*/", a ako se on ne pojavi, kod biva iskomentiran do samoga kraja.

Test primjeri

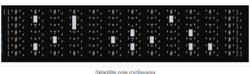
Detalji svih test primjera dani su u samim kodovima te se čitatelja upućuje da se konzultira s njima u slučaju nejasnoća, ali također i da pokretanjem svakog od primjera te njihovom pažljivom analizom ustvrdi slijed izvršavanja naredbi. Program crta 2D grafički prikaz skladišta nakon svake izvršene naredbe te usto i ispisuje tip naredbe i uspješnost njena izvođenja.

Test primjer 1

Prvi test primjer započinje funkcijom `fakt` koja prima nenegativni cijeli broj i vraća njegov faktorijel. Razlog korištenja ove funkcije je demonstracija valjanog rada rekurzija u konstruiranom jeziku, ali isto tako i za kasniju upotrebu da bude vidljivo da se AST-ovi konstruiraju kako trebaju te da kao argumente određenih funkcija možemo slati proizvoljno složene aritmetičke izraze.

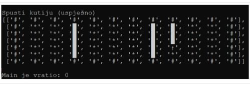
Unutar funkcije `main`, robot obavlja svoje osnovne funkcionalnosti, odnosno kreće se po dopuštenim mjestima u skladištu, popravlja sve električne kvarove te prenosi kutije i na kraju ih posloži u slova IP jer znamo da, osim predstavljanja kvalitete koda pred Nadzornim odborom tvrtke **fsociety**, ujedno mora i ostvariti maksimalni broj bodova iz kolegija *Interpretacija programa*. Funkcija `main` vraća cijeli broj 0 kao znak da je program izvršen uspješno. Cijeli broj unutar danog jezika odgovara tipu podatka `Number`.

Konačni numerički izlaz programa i stanje skladišta prije i nakon njegova izvršavanja dani su na sljedećoj slici:



```
[['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'],
 ['#', '█', '*', '█', '*', '*', '*', '*', '█', '*', '*', '#'],
 ['#', 'T', '*', '█', '*', '*', '↑', 'T', '*', '*', 'T', '#'],
 ['#', '*', 'T', '*', '*', '*', '█', '*', '*', '█', '*', '#'],
 ['#', '*', '█', '*', '*', '█', 'T', '█', '*', '*', '*', '#'],
 ['#', '█', '*', '*', '*', '*', '█', '*', '*', '█', '█', '#'],
 ['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#']]
```

Skladište prije izvršavanja



```
Spusti kutiju (uspješno)
[['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'],
 ['#', '*', '*', '█', '*', '*', '*', '█', '█', '*', '*', '#'],
 ['#', '*', '*', '█', '*', '*', '*', '█', '█', '*', '*', '#'],
 ['#', '*', '*', '█', '*', '*', '*', '█', '█', '*', '*', '#'],
 ['#', '*', '*', '█', '*', '*', '*', '█', '↑', '*', '*', '#'],
 ['#', '*', '*', '█', '*', '*', '*', '█', '*', '*', '*', '#'],
 ['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#']]

Main je vratio: 0
```

Skladište nakon izvršavanja

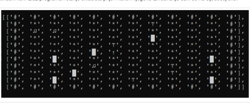
Test primjer 2

Drugi test primjer sastoji se samo od funkcije `main` koja za povratni tip ima `List`, također definiran od strane autora jezika. Temeljna ideja ovoga primjera je suptilno i polagano prolaženje kroz skladište i pritom izbjegavajući kritične razine zvuka koje bi oštetile robota, a odgovaraju 70 dB za slobodnog robota, odnosno 50 dB kada nosi kutiju.

Tijek koda odgovara standardnom načinu na koji bi osoba koja ne zna gdje su opasni izvori zvuka analizirala skladište. Naime, upravljač robota ima pregled skladišta te vidi njegove kutije i kvarove, no ne i izvore zvuka te se zbog toga po skladištu treba gibati polako, navodeći se senzorom za intenzitet zvuka koji robot ima. Također treba naći prikladan put kroz koji će kutije prenositi jer, iako je možda samostalno uspio proći kraj izvora zvuka od 50 ili više dB, zna da to ne može napraviti noseći kutije.

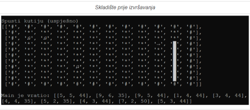
Imajući na umu sve navedeno, korisnik robota navodi uz uske prolaze, točno dovoljne da ga snažan zvuk ne ošteti i na taj način se sve kutije raspoređene po cijelom skladištu prenosi na jedan njegov kraj, daleko od opasnih izvora.

Konačni lisni izlaz programa i stanje skladišta prije i nakon njegova izvršavanja dani su na sljedećoj slici:



```
[['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'],
 ['#', '*', '*', '*', '*', '*', '*', 'T', '*', '*', '*', '#'],
 ['#', '♫', '♫', '*', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '█', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '*', 'T', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '█', '↑', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '█', '*', '*', '*', '*', '*', '*', '*', '█', '#'],
 ['#', '*', '*', '*', '♫', '*', '*', '*', 'T', '*', '*', '#'],
 ['#', '*', '*', '█', '*', '*', '*', '*', 'T', '*', '*', '#'],
 ['#', '*', '█', '*', '*', '*', 'T', '*', '*', '*', '█', '#'],
 ['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#']]
```

Skladište prije izvršavanja



```
Spusti kutiju (uspješno)
[['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '♫', '♫', '*', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '→', '█', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '█', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '█', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '█', '#'],
 ['#', '*', '*', '*', '♫', '*', '*', '*', '*', '*', '█', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '█', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '█', '#'],
 ['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#']]

Main je vratio: [[5, 5, 44], [9, 6, 35], [9, 5, 44], [1, 4, 44], [3, 4, 44],
[4, 4, 35], [5, 2, 35], [4, 3, 44], [7, 2, 50], [5, 3, 44]]
```

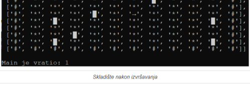
Skladište nakon izvršavanja

Test primjer 3

Treći i posljednji test primjer implementira složenu rekurzivnu funkciju `spirala` korisnički definiranog povratnog tipa `Bool` koji je ekspresivan na razini Kleenejeve trovaljane logike i to s logičkim konstantama `True`, `Unkwnown` i `False` te standardnim logičkim veznicima.

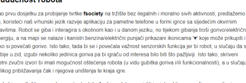
Funckija `main` također je tipa `Bool` i to tako da joj je povratna vrijednost istinita ako je robot uspio pronaći dovoljno praznog mjesta za blok zadane veličine. Pretraživanje se vrši tako da robot zadni blok iscrtava rekurzivno spiralno te ako spiralu uspije popuniti, kroz povratnu vrijednost pošalje upravljaču signal da mjesta ima. Očito je ovaj pristup koristan kada upravljač nema potpuni pristup skladištu iz bilo kojeg razloga. Naravno, uspješnost robotova pretraživanja s jednostavnijim kodom odgovara strukturi skladišta koje je u primjeru dano, za općenito skladište treba dodati i mogućnost zaobilaska prepreka te traženja nakon njih ako se nalaze u istom redu.

Konačni istinitosni izlaz programa i stanje skladišta prije i nakon njegova izvršavanja dani su na sljedećoj slici:



```
[['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'],
 ['#', '←', '*', '*', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '█', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '█', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '█', '*', '*', '*', '*', '*', '*', '*', '█', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '*', '█', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '█', '*', '*', '*', '*', '*', '*', '*', '█', '#'],
 ['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#']]
```

Skladište prije izvršavanja



```
Okreni se za 180 stupnjeva (uspješno)
[['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '█', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '*', '*', '█', '*', '→', '*', '*', '*', '*', '#'],
 ['#', '*', '█', '*', '*', '*', '*', '*', '*', '*', '█', '#'],
 ['#', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '*', '█', '*', '*', '*', '*', '*', '*', '*', '#'],
 ['#', '*', '█', '*', '*', '*', '*', '*', '*', '*', '█', '#'],
 ['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#']]

Main je vratio: 1
```

Skladište nakon izvršavanja

Budućnost robota

Kao prvu dosjetku za probijanje tvrtke **fsociety** na tržište bez ilegalnih i moralno sivih aktivnosti, predlažemo da, koristeći naš vrhunski jezik razvije aplikaciju za pametne telefone u formi igrice sa sljedećim okvirnim pravilima. Robot se giba i interagira s okolinom kao i u danom jeziku, no tijekom gibanja troši gorivo/električnu energiju, a na mapi se nalaze i kanistri benzina/električni punjači prikazani ikonicama '♥' koje može prikupiti i tako si povećati gorivo. Isto tako, tada bi se i povećala važnost senzorskih funkcija jer bi robot, u slučaju da se zabije u zid, izgubi nekoliko jedinica goriva pa bi igraču od interesa bilo biti što pažljiviji. Isto tako, skriveni štetni zvučni izvori bi imali mogućnost oštećenja robota (u vidu gubitka goriva i/ili funkcionalnosti), a u slučaju velikog približavanja čak i njegova uništenja te kraja igre.

Kada bi se odlučilo na ovakvu varijantu, također bi se dalo razmisliti i o interaktivnom načinu upravljanja robotom. U danoj verziji jezika, korisnik mora unijeti sve naredbe odjednom—iako se čini da ovo smanjuje ekspresivnu moć jezika, nije tako. Naime, ovome se lako doskoči tako da korisnik, unoseći naredbu po naredbu, pokreće program po program te ga tako gradi do njegove konačne verzije koja [mu,joj] je zadovoljavajuća. S druge strane, sučelje prikladnije za masovnu upotrebu više bi nalikovalo na opisano, odnosno pružalo mogućnost izvršavanja pojedinih naredbi. Isto tako, u trenutni jezak bi se uz nekoliko slova (konkretno `case=False` u nekim slanjima argumenata) moglo omogućiti da umjesto npr. naredbe `forward` jezik prihvaća i naredbe poput `fOrWaRd`, no kako je jezik osmišljen za iskusnu grupu hakera, smatramo da ovakva pojednostavljenja nisu bila potrebna.

Konačno, autori smatraju da se proširenjem mogućnosti lisnog tipa u kodu može implementirati pretraživanje u dubinu i širinu te da robot na taj način može tražiti putove, dostupni prostor i slično i efikasnije obavljati svoje zadaće.