Datorövningar i ändliga automater och reguljära språk

Ladda ner filen `automata.py`. Den innehåller Python-kod för två automater som finns i DM-boken.

Syfte

Den här laborationen handlar om automater och reguljära uttryck. Du får arbeta med olika typer av automater, och i synnerhet får skapa en automat som bestämmer beteendet hos en simulerad sköldpadda.

Du får också pröva på Pythons reguljära uttryck, som är väldigt användbara för att söka information i text. De flesta andra programmeringsspråk har också stöd för reguljära uttryck.

Accepterande automater

Den första automaten i filen är den accepterande automaten på sidan 249 i DM-boken. Det finns två funktioner. Den första heter `auto1_run` och tar en sträng av 0:or och 1:or och kör genom automaten, tecken för tecken. Funktionen `auto1_run` anropar funktionen `auto1_step` för varje tecken i strängen och håller också reda på det nuvarande tillståndet (i början är det 's0'). När alla tecken har gåtts igenom testar `auto1_run` om det sista tillståndet är ett accepterande tillstånd. Den returnerar `True` om automaten hamnar i ett accepterande tillstånd (s1), och `False` annars. Den skriver också ut vilka tillstånd som besöks.

Det egentliga arbetet görs i funktionen `auto1_step` som tar det nuvarande tillståndet för automaten ('s0' eller 's1') och ett tecken ('0' eller '1') och returnerar nästa tillstånd. Funktionen är uppbyggd av två nivåer villkorssatser. Den första nivån tittar på vilket tillståndet är och andra nivån tittar på vilken inmatning som kommer. Sedan returneras nästa tillstånd (dvs det man når om man följer pilen i grafen). T ex säger de allra första raderna att tillstånd 's0' och indata '0' leder till tillstånd 's0'.

Uppgift 1: Testa automaten för ett antal strängar: '0','1', '01, '10', '0001', '00010', '01010101'. Du anropar så här: `auto1_run('0')` Tänk igenom vad som händer.

Mealy-automater

Den andra automaten som finns i filen är en Mealy-automat som du ser på sidan 245 i DM-boken. Den består av två funktioner, `mealy_run` och `mealy_step`. Den senare returnerar både en utdata ('0' eller '1') och ett nytt tillstånd.

Uppgift 2: Testa `mealy_run` på strängarna: '0', '1', '1101', '001010100', '111111' och tänk på vad som händer.

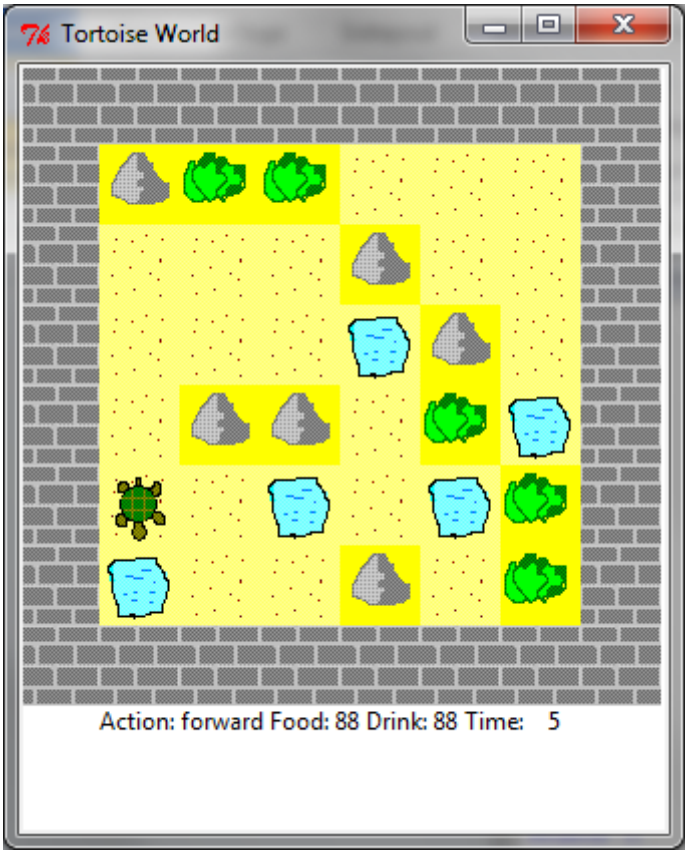
Sköldpaddans värld

Ladda ner filen `Tortoise v2.zip` och extrahera filerna i den. Här i hittar du en Python-fil (`tortoise.py`) och ett antal bildfiler (av typen GIF).

Öppna och titta i filen `tortoise.py`. Här hittar du det följande:

- Ett antal import-kommandon.
- En klass för ett grafiskt användargränssnitt (GameInterface)
- En funktion tortoiseworld som är en simulering av en sköldpadda.
- En funktion call_automata som är en ändlig tillståndsautomat som styr sköldpaddans beteende. För närvarande är sköldpaddan inte särskilt klok. Din uppgift är att göra den klokare, genom att förbättra tillståndsautomaten.

Pröva först att köra filen (tryck F5). Ett litet fönster öppnas. Det kan råka hamna bakom något annat fönster, men du bör kunna se det markeras längst ner på skärmen. Fönstret föreställer ett inhägnat sandigt område med stenblock, sallad och vattenpölar. I övre vänstra hörnet ser du också en sköldpadda. Sköldpaddan gör inte särkilt mycket just nu. I fönstrets nedre del ser du vilken handling sköldpaddan just har gjort, vilket tillstånd automaten befinner sig i, sköldpaddans mat- och vätskenivåer (som tickar ner) och hur mycket tid som har gått sedan simuleringen startade. Efter ett tag går mat- och vätskenivåerna ner under 0, och då dör den stackar sköldpaddan.



Automaten

Funktionen som implementerar sköldpaddans automat ser ut så här:

```
def call_automata(state,free_ahead,lettuce_ahead,lettuce_here,
water_ahead,water_here,hungry,thirsty):
    if state==0:
        if hungry and lettuce_here:
            return 0,'eat'
        elif thirsty and water_here:
            return 0,'drink'
        else:
            return 0,'wait'
```

Den tar som argument det nuvarande tillståndet (state), och ett antal parametrar som tillsammans utgör automatens inmatning.

Inmatning

Imatningsparametrarna är:

- free_ahead – platsen framför sköldpaddan är ej blockerad
- lettuce_ahead - det finns en salladsplanta på platsen framför sköldpaddan
- lettuce_here – det finns en salladsplanta där sköldpaddan är
- water_ahead – det finns en vattenpöl framför sköldpaddan
- water_here – det finns vatten där sköldpaddan är
- hungry - matnivån < 50
- thirsty - vätskenivån < 50

Alltså, vid varje tidssteg får automaten en inmatning som består av dessa parametrar tillsammans. De olika parametrarna har booleska värden (True eller False).

Nytt tillstånd och utmatning

Automaten returnerar två saker:

- automatens nya tillstånd, som kommer att ges till automaten nästa gång
- sköldpaddans nästa handling

Tillståndet är 0 i början.

Handlingarna består av strängar:

- 'eat' – sköldpaddan äter sallad, om det finns där den befinner sig just nu. Matnivån blir 100.
- 'drink' - sköldpaddan dricker vatten, om det finns där den befinner sig just nu. Vätskenivån blir 100.
- 'left' - skölpaddan vänder sig 90 grader åt vänster, dvs moturs
- 'right' - skölpaddan vänder sig 90 grader åt höger, dvs medurs
- 'forward' – sköldpaddan tar ett steg framåt
- 'wait' – sköldpaddan gör ingenting

All rörelse förbrukar två enheter mat och dryck. Att vänta, äta eller dricka förbrukar en enhet.

Uppgift 3

Din uppgift är att förbättra automaten, så att sköldpaddan rör sig så att den hittar sallad och vatten, och på så sätt kan överleva längre. Börja gärna med att försöka rita en bättre automat och koda den sedan i funktionen call_automata

Här är några tips på hur du kan förbättra automaten.

- Sköldpaddan rör aldrig på sig. Den behöver flytta på sig (’forward’) för att kunna hitta sallad och vatten.
- Om sköldpaddan stöter på en vägg eller sten (not free_ahead) så bör den svänga (’left’ eller ’right’).
- Om sköldpaddan träffar på sallad eller vatten så kan den äta (’eat’) respektive dricka (’drink’).
- Sköldpaddan ser bara saker som är framför den. Därför kan det vara bra att den ibland vänder sig och tittar åt sidorna också, när den letar efter mat/vatten.
- Med den nuvarande automaten så dör sköldpaddan efter 100 sekunder. Om du lyckas få sköldpaddan att leva längre så har du lyckats. Obs! Eftersom sallad, vatten och stenar placeras ut slumpmässigt, så kan sköldpaddan klara sig olika bra från gång till gång.

Obs: du ska bara använda en automat, som ska ha formen:

```
if state==tillstånd:  # Tillstånd 1
    if test_med_input_parametrar:   # Övergång
        return nytt_tillstånd, handling
    elif test_med_input_parametrar: # Övergång
        return nytt_tillstånd,  handling
    ...
    else:                                          # Övergång
        return nytt_tillstånd, handling
elif state== tillstånd:  # Tillstånd 2
    if test_med_input_parametrar:   # Övergång
        return nytt_tillstånd, handling
     ...
...
```

Du ska inte använda slump-funktioner, t ex random.choice, i din automat.

Man klarar sig ganska långt med en automat med bara ett enda tillstånd, men det finns ett antal tillfällen då det är bra att ha flera tillstånd. T ex:

- Om man vill att sköldpaddan varannan gång ska svänga till vänster och varannan gång till höger, då dess väg är blockerad, kan olika tillstånd användas för att hålla reda på vilket håll sköldpaddan ska svänga nästa tillfälle då den blir blockerad. Vid varje tillfälle ska naturligtvis sköldpaddan fortsätta svänga åt ett håll tills det är fritt framåt. Det är först när den har gått framåt och stöter på ett nytt hinder som den ska byta håll.
- Om sköldpaddan ska titta till vänster och höger och sedan vända sig rakt fram igen, så kan olika tillstånd användas för att hålla reda på var i 'vänster-höger-fram'-sekvensen som sköldpaddan befinner sig i just nu.

Reguljära uttryck

Python har en modul `re` för reguljära uttryck. Reguljära uttryck kan användas för att söka i en sträng, och det finns en hel del kraftfulla konstruktioner. Så här kan vi t ex göra för att testa om en sträng börjar med en följd (1 eller fler) av 0:or och därefter en 1:a:

```
>>> import re
>>> re.match('0(0*)1','000102')
```

Syntaxen för reguljära uttryck skiljer sig något från den i DM-boken. Här är några av de viktigaste konstruktionerna:

- `.` matchar ett tecken (vilket som helst)
- *r*`*` Kleenestjärna: matchar 0 eller flera upprepningar av uttrycket *r*.
- *r*`+` matchar 1 eller flera upprepningar av uttrycket *r* (ej samma som + i boken).
- *r*{*m*,*n*} matchar mella *m* och *n* upprepningar av r.
- `[a-z]` matchar ett tecken mellan a och z. Du kan byta ut a-z mot vilka tecken du vill.
- `[abcd]` matchar ett av tecknen a,b,c,d. Du kan byta ut a,b,c,d mot vilka tecken du vill.
- `[^a-z]` och `[^abcd]` matchar komplementet till de angivna tecknen.
- *r1*`|`*r2* matchar någon av *r1* och *r2*, dvs unionen (+ i DM-boken). Du kan också ha fler *r*.
- Sammansättningen (punkten i DM-boken) får man helt enkelt genom att skriva två uttryck efter varandra.
- Parenteser kan användas för att avgränsa ett uttryck.
- Om du vill ha med ett specialtecken som `.`, `*` eller `+` som vanligt tecken så sätter du `\` framför.
- `^` och `$` matchar början respektive slutet på strängen.
- Mellanslag räknas också som tecken. 'a b' matchar alltså inte 'ab'.

Här är några exempel:

- `'[a-d]*e'` matchar strängar som börjar med 0 eller flera upprepningar av a,b,c och d, och sedan kommer ett e. T ex: acbe, ce, e
- `'([a-d]+)|([m-p]+)'` matchar strängar av antingen en eller flera upprepningar av a,b,c,d eller av m,n,o,p. T ex: ababa eller mop.

Det finns ett antal funktioner som jobbar med reguljära uttryck, och bl a kan söka i början av en sträng eller var som helst i en sträng. Vi använder funktionen `re.match(`*`uttryck, sträng`*`)` som söker efter ett reguljärt uttryck i början av en sträng. T ex:

```
>>> re.match('0*1$','0001')
<_sre.SRE_Match object at 0x00E05CD0>
>>> re.match('0*1$','0002')
```

Om funktionen inte lyckas matcha uttrycket mot strängen så returnerar den ingenting (`None`, skrivs inte ut), vilket sker vid andra anropet ovan. Om matchningen lyckas, returnerar funktionen ett match-objekt, som innehåller information om hur matchningen lyckades.

Uttrycket `'0*1$'` beskriver strängar som börjar med 0 eller fler nollor, sedan en etta, och sedan tar strängen slut (`$`). Vi har med `$` för att vi vill matcha hela strängen, inte bara början. Det ska du också göra i samtliga uppgifter här nedan.

Om du vill söka var som helst i strängen, använder du funktionen `re.search` i stället, med samma sorts argument.

Vi kan nu enkelt testa om en sträng tillhör ett reguljärt språk genom att formulera ett reguljärt uttryck som beskriver språket, och sedan använda `re.match`. Vi måste alltid komma ihåg att lägga till `$` på slutet för att vara säkra på att hela strängen matchar. Ta exempel 9.3 i DM-boken. Det språket kan beskrivas med `'((auto|flyg)(mat|pilot))$'`.

Uppgift 4: Testa om strängarna `'auto'`, `'autopilot'`, `'autopiloten'`, och `'läskautomat'` tillhör språket i exempel 9.3.

Uppgift 5: I operativsystem som Linux andvänder man ofta reguljära uttryck för att söka efter filer. Skriv reguljära uttryck som testar om en sträng (dvs ett filnamn) (a) har extensionen '.tex', (b) saknar extension, (c) har tre avslutande siffror innan extensionen (d) har någon av extensionerna .txt, .doc, .tex, .ini. Testa de olika reguljära uttrycken för a-d på dessa filnamn: `myfile.tex`, `yourfile`, `cooldoc123.tex`, `textdocument.doc`, `filebile.ini`.

Uppgift 6: Beskriv språket för alla naturliga tal (exempel 9.5 i DM-boken) som ett reguljärt uttryck och testa det på ett antal strängar: `'0'`,`'0a'`, `'04'`, `'15'`, `'1066'`, `'femton'`.

Uppgift 7: Gör samma sak med språket av räkneuttryck (övning 9.26 i DM-boken). Tänk på att skriva plus och gånger som `\+` och `\*` i det reguljära uttrycket, eftersom + och `*` används som specialtecken. Testa på: `'0'`, `'10'`, `'10+20'`, `'7*8+33/5-999'`, `'mycket'`, `'22**13+-4'`, `'+32'`, `'15+'`.

Redovisning

Lösningarna för samtliga uppgifter visas för assistenten och redovisas skriftligt. Om ni är två personer som har arbetat ihop, så ska ni ange det i redovisningen. Ni får gärna diskutera med andra personer/grupper under laborationen, men varje grupp ska lämna in sin egen lösning.

Checklista

[ ] Sköldpaddan överlever mer än 100 s.

[ ] Samtliga uppgifter (1 till 7) visade för assistenten

[ ] Samtliga uppgifter (1 till 7) redovisade i fil.

[ ] Era namn står i en kommentar i början av filen.