Datorövningar i logik

Syfte

Den här laborationen ger övning i att tolka logiska formler, och att bestämma hur de blir sanna. Den tar upp både satslogi där du får tilldela sanningsvärden till satser, och predikatlogik, och predikatlogik där du får skapa tolkningar som gör formlerna sanna.

Satslogik

Ladda ner modulen `logic.py` och spara den med dina andra Python-filer. Du kan nu importera den till dina filer genom att lägga till i början:

```
from logic import *
```

Med hjälp av den här modulen kan du skapa logiska formler, t ex:

```
>>> f = A & ~B
```

Detta var en satslogisk formel, och betyder "A och inte B". Den inledande snutten 'f = ' är inte del av själva formeln, utan innebär att variabeln f tilldelas formeln ' A & ~B'. Det är själva formeln det är frågan om här, och inte formelns sanningsvärde.

Vi ska nu skapa en satslogisk tolkning som talar om vilka (atomära) satser som är sanna:

```
>>> i = [A]
```

Här är satsen A sann, och satsen B (som inte är med) är falsk. Generellt fungerar det så här:

- Om en sats ska vara sann så ska den finnas med i listan.
- Satser som inte är med räknas alltid som falska.

En tolkning motsvarar alltså en rad i sanningsvärdestabellen: varje atomär sats har ett bestämt sanningsvärde.Om vi hade skrivit tolkningen som en rad i en sanningsvärdestabell hade den alltså sett ut så här:

A	B
1	0

Prova att beräkna sanningsvärdet för formeln `f` i tolkningen `i`:

```
>>> val(f,i)
True
```

Alltså, om `A` är sann och `B` är falsk så blir `A & ~B` sann. I tabellform skulle det sett ut så här:

A	B	A & ~B
1	0	1

Pröva också en tolkning där både A och B falska:

```
>>> val(f, [])
False
```

Här är samtliga logiska konnektiv i modulen:

- `~A` "inte A"
- `A & B` "A och B"
- `A | B` "A eller B"
- `A > B` "om A så B"
- `A == B` "A om och endast om B"

(Tyvärr är det ont om tangentbord med logiska symboler på, så vi får hålla till godo med de här varianterna).

Ni ska nu försöka hitta tolkningar som gör olika satslogiska formler sanna. Tag t ex

```
f = A | C
```

dvs A eller C. Den kan göras sann antingen genom att göra A sann eller C sann.
Om vi nu beräknar värdet för denna formel när A är sann och C är falsk så får vi:

```
>>> val(f,[A])
True
```

Uppgift 1
Gör samma sak för följande formler:

1. `(rainy | windy)`
2. `rainy > windy`
3. `(rainy > wet) & (~rainy > ~wet)`
4. `(rainy | windy) & (rainy > wet) & (windy > cold)`
5. `((rainy & windy) == snow) & winter & (~snow | ~winter)`
6. `(rainy == windy) & (winter | summer) & (spring | summer) & (~winter > ~windy)`
7. `(wet > sunny) & (summer | spring) & (snow == winter)`

Uppgift 2

Försök uttrycka följande som satslogiska formler och testa varje formel i olika tolkningar för att se om den fungerar.

1. Det är antingen sommar (summer), vinter (winter), vår (spring) eller höst (autumn), men aldrig två av dem samtidigt. (Detta kallas för 'exkusivt eller'.) Testa med tolkningar där det är ingen årstid, en årstid, eller två årstider, för att se om det fungerar.
2. Det snöar (snow) bara på vintern, och det kan aldrig regna (rainy) och snöa samtidigt.
3. När det snöar så är det kallt (cold), och när det regnar så är det inte kallt.

Obs! Du har bara tillgång till ett antal fördefinierade satser - om du t ex försöker skriva `(cloudy > hurricane)` så skulle inte det fungera eftersom satserna 'cloudy' och 'hurricane inte finns definierade.

Redovisning uppgift 1 och 2

Lös uppgifterna genom att skapa en Python-fil enligt följande mönster:

```
from logic import *

print('Satslogik')
f1 = (rainy | windy)
print('f1:', val(f1,[...]))
```

och så vidare för de andra formlerna...

När du sedan kör filen ska du få en utskrift i stil med:

```
Satslogik
f1: True
```

och så vidare...

Predikatlogik

Predikatlogiken har samma konnektiv som satslogiken, men har dessutom följande saker:

- Ett antal predikat: `happy, sad, hungry, loves, hates, knows, human, dog, cat, owns, male, female, brother, sister, friend`.
- Ett antal namn på individer: `john, mary, bob, alice, fido, cleo, omalley, fluffy`
- Ett antal satslogiska variabler: `u, v, x, y, z`.
- De två kvantifikatorerna `forall(`*`variabel`*`,`*`formel`*`)` ("alla") och `exists(`*`variabel`*`,`*`formel`*`)` ("någon").
- Likhet `==` mellan namn/variabler (samma symbol som för ekvivalens, men det senare är mellan satser).

Vi kan nu skapa predikatlogiska formler:

```
>>> f1 = dog(fido) & hungry(fido)
>>> f2 = forall(x,happy(x))
```

För att skapa en tolkning för predikatlogiska formler måste vi ha två saker: de atomära satser som är sanna, och de individer som finns (domänen). Här är ett exempel:

```
>>> val(f1, [dog(fido),cat(cleo),hungry(fido),happy(cleo)], [fido, cleo])
True
```

Detta ger en tolkning där det finns två individer: Fido och Cleo. Om dessa vet vi att Fido är en hund, Cleo är en katt, Fido är hungrig och Cleo är lycklig.Vi vet också att ett antal (enkla) satser är falska eftersom de är utelämnade: Fido är inte en katt och ej heller lycklig, och Cleo är inte en hund och ej heller hungrig. I tabellform ser tolkningen ut så här:

Individer	{ cleo, fido }
cat(cleo)	1
cat(fido)	0
dog(cleo)	0
dog(fido)	1
happy(cleo)	1
happy(fido)	0
hungry(cleo)	0
hungry(fido)	1

Obs! Om vi hade haft fler individer (t ex fluffy) så hade vi fått fler enkla satser att tilldela sanningsvärden. Vi har inte heller brytt oss om att skriva ut de övriga predikaten här, t ex loves.

Uppgift 3

Tag nu följande predikatlogiska formler och försök hitta tolkningar för dem:

1. `forall(x, hungry(x) > ~happy(x))`
2. `exists(x, hungry(x) & ~happy(x))`
3. `~hungry(fido) & forall(x,~hungry(x) > happy(x))`
4. `hungry(fido) & happy(cleo) & forall(x,~hungry(x) > happy(x))`
5. `exists(x, hungry(x) & ~happy(x) & ~(x == fido))`
6. `exists(x, loves(x,john) & loves(john,x))`
7. `exists(x, loves(x,john)) & forall(x,forall(y, (loves(x,y) & loves(y,x)) > (happy(x) & happy(y))))`
8. `loves(john, mary) & hates(mary,john) & forall(x, exists(y, (loves(x,y) & hates(y,x)) > ~happy(x)))`
9. `human(john) & human(bob) & forall(x,human(x) > exists(y, (dog(y) | cat(y)) & owns(x,y)))`
10. `exists(x,dog(x)) & exists(y,cat(y)) & exists(z,female(z)) & exists(u,male(u)) & forall(x,male(x) | female(x))`

Använd val-funktionen som beskrivs ovan för att visa att tolkningen stämmer (ger True). Lägg gärna varje anrop i en print-sats så att alla resultat skrivs ut när Python-filen exekveras.

Uppgift 4

Man brukar skriva "någon hund är hungrig" som
`exists(x, dog(x) & hungry(x))`
och "alla hundar är hungriga" som
`forall(x, dog(x) > hungry(x)).`

Varför skriver man inte
`exists(x, dog(x) > hungry(x))` och `forall(x, dog(x) & hungry(x))`?

Undersök saken genom att ta fram tolkningar där de senare varianterna inte ger önskat resultat.

Uppgift 5

Dags för ett litet relationsdrama. Skapa följande tolkning (i,d):

```
i=[knows(bob,john),knows(john,bob),knows(mary,alice),knows(alice,mary)]
d=[bob,john,mary,alice]
```

Predikatet `knows` är en relation. Skriv predikatlogiska formler som testar om `knows` är:

1. reflexiv
2. symmetrisk
3. transitiv

Testa också formlerna på tolkningen (i,d).

Testa samma egenskaper på relationen `loves` i följande tolkning:

```
i = [loves(bob,mary),loves(john,john),loves(mary,alice),loves(bob,alice)]
d = [bob,john,mary,alice]
```

Redovisning uppgift 3, 4 och 5

Uppgifterna i predikatlogik redovisas på samma sätt som dem i satslogik.

Allmänt om redovisning

Övningarna 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

[ ] Samtliga uppgifter (1, 2, 3, 4, 5) visade för assistenten

[ ] Samtliga uppgifter (1, 2, 3, 4, 5) redovisade i fil.

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