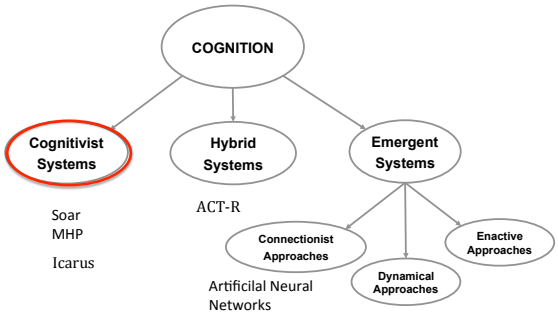
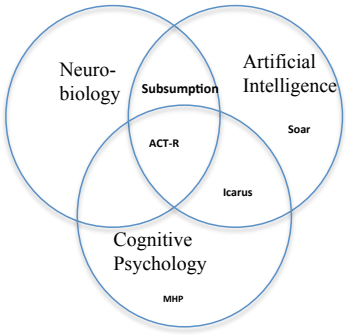
Løsningsforslag eksamen TDT4137 H17

Oppgave 1 (20%)

a) To figurer: en basert på cognition-begrepet og en fra faginspirasjon. Systemer (3 av disse): Icarus, Soar, ACT-R, MHP, Brooks subsumption, (Nevrale nett):





faginspirert

b) **MHP-subsystemer**: **perseptuelt**-, **kognitivt**- og **motor**-system med tilhørende perseptuell-, kognitiv- og motor-prosessor.

Basisoperasjoner kognitiv prosessor: **match** (fysisk, navn, klasse), **hente fra LTM** (navn, klasse), og **generere motorkommando**.

Tid til en oppdager at symbolene II og 2 er samme tall: Antar at navnet (tallverdi) allerede ligger som en semantisk representasjon av II i WM (og visuell) ved T =0. Når symbolet "2" dukker opp, må det inn i VIS og WM (1 syklus perseptuell prosessor), så brukes visuell representasjon til å hente navnet/tallverdien fra LTM (1 syklus kognitiv prosessor). Nå kan de to navnene i WM matches og gi resultat lik/ulik (1 syklus kognitiv prosessor). Tiden blir da

$\mathbf{T = T_p + 2\ T_c}$ som hvis en setter inn gir

= 100+ 2*70 = 240 ms (men ikke krav om å huske prosessortider)

Kan gjerne tegne opp hvis en vil…(men ikke krav). Ikke trekk om en også generer en motorkommando …men det er ikke spurt om å trykke ned noe hvis tallene er like f.eks)

c) Hicks lov er en versjon av "**Usikkerhetsprinsippet**" (P7- "Princple of Uncertainty"). Beslutningstiden til den kognitive prosessor øker med usikkerheten i vurderingen eller beslutningen som skal gjøres. H er den informasjonsteoretiske entropi og I_c er en konstant. Når n (antall valg/stimulus) øker, så øker beslutningstiden.

d) **Analogi-resonnering**?: " evnen til å oppfatte og bruke **relasjonell** similaritet (også kalt **strukturell similaritet)** mellom to situasjoner eller hendelser

Hvordan representerer **Gentner kunnskap** for å kunne mappe target og base analogi-resonneringen? : ved **objects**, object **attributes**, og **relations** mellom objects. Kunnskapen representeres ved **propositional networks of nodes and predicates**.

Forklar kort "**The Systematicity Principle**": "A predicate that belongs to a mappable system **of mutally interconnecting relationships**, is more likely to be imported into target than is an isolated predicate" (Gentner).

Thus **preference** for matching **deep systems** of connected relations, rather than smaller relational sets: i.e. **Higher-order relations** and systems of relations are preferred.

Oppgave 2 (20%)

a) **Virkemåte nevron**: netto-input (summering m/threshold)+ aktivering.
(tegn gjerne opp)
Ligning ved sigmoid aktivering: Y= sigmoid(..)
Ulike aktiveringsfunksjon som f.eks : step, sign, sigmoid, tanh, ReLU

Analogi kunstig-biologisk:
Input ->dendritter, nouron->soma, vekter ->synapser og output->axoner
Backprop?

b) Fire trinn se "pseudo"-kode nedenfor i rødt: **Initaliser, Aktiver, Oppdater** vekter (oppgitt delta regel), og **sjekk konvergens**. Koden oppdaterer også threshold (θ) som w0, men også OK om ikke θ endres. α er **læringshastigheten** (kan evt. forklare litt her).

```
# Perceptron learning algorithm i (Python syntax)
#Training data on the format: ((x0,x1,x2),y) where x0 is related to bias (allways -1)
trainingdata = [((-1,0, 0), 0), ((-1, 0, 1), 0), ((-1, 1, 0), 0), ((-1, 1, 1), 1)] # Example training data
for AND (similar line for OR)
# Learning rate alpha = 0.1
# Step 1 - Initialise input weights (random) start value (w0,w1,w2) - where w0 is threshold theta (bias)
wi = [0.3, -0.1, -0.5]
#
#
#-------------------------------------------------
#  xwsum :computes net weighted input
#  xs - input data
#  wi - weights
#-------------------------------------------------
def xwsum(xs, wi):
     return sum(x * w for x, w in zip(xs, wi))
#
#  Main:
#
# Do weight training until convergence
while True:
    error_count = 0
    for xi, yd in trainingdata:
         # Step 2 -  Do activation (use given step function)
        if xwsum(xi, wi) >= 0:
            y = 1
        else:
            y = 0

        # calculate error (desired - output)
        e = yd - y

        if e != 0 :
            error_count += 1

         # Step 3 -  Update weights according to delta rule
            for i, x in enumerate(xi):
                wi[i] += alpha  * x * e    // endr vekt

    # Step 4 -  Convergence?
    if error_count == 0:           // sjekk om fremdeles feil
        break
```

Men det er her også greit å skrive mer høy-nivå pseudokode, ser jo om detaljene i algoritmen er plass når en skal regne endring i vekter nedenfor).

Perseptronet vil **ikke** være i stand til å lære datasettet vårt, siden dataene **ikke** er **lineært separabelt** (ingen rett linje som kan skille de to kategoriene 0 og 1).

Feilen ∈ (egentlig uten i her siden perseptron) defineres som (se kode over):

$e = y_d - y$ gir feilen (y_d desired output og y er det vi får ut)

Aktivering:

$y = \text{step}(x1*w1+x2*w2- \theta) = \text{step}(0.2*0.2+0.3*0.1 - 0.2)$
$= \text{step}(-0.13) = 0$

regner så ut feil

$e = y_d - y = 1-0 = 1$

Setter inn i oppgitt deltaregel som gir følgende endringer i vekter:

$\Delta w_1(p) = \alpha\, x_1(p)\, e \quad (p) = 0.1 * 0.2 * 1 = 0.02$
$\Delta w_2(p) = \alpha\, x_2(p)\, e \quad (p) = 0.1 * 0.3 * 1 = 0.03$

$w_1(p+1) = w_1(p) + \Delta w_1(p) =$ 0.2+0.02= **0.22**
$w_2(p+1) = w_2(p) + \Delta w_2(p) =$ 0.1 +0.03 = **0.13**

Threshold θ, dvs w0, kan evt. også oppdateres på samme måte (kreves ikke):

$\Delta w_0(p) = \alpha\, x_0(p)\, e \quad (p) = 0.1 * -1 * 1 = -0.01$
$w_0(p+1) = w_0(p) + \Delta w_0(p) =$ 0.2-0.01= **0.19**

c) **CNN** er en type nettverk hvor input til er inspirert fra nevrofysiologi/nevrovitenskap og angir at et nevron bare ser mindre lokale felter/rektangler i input-bilde(**Local receptive field**). Dersom alle input-nevronene skulle ta inn hele bilde (**fully-connected** neural network) vil det gi store mengder av parametre. Ved bruk av ideen med Local receptive field blir **antall parametre mye mindre**.

Et konvolusjons-lag øker det totale antall av features. Hver konvolusjonskjerne (vektene) prøver å lære forskjellige translasjonsinvariante features og gjenbrukes/"kopieres" bortover hele bildet. Konvolusjon er en lineær operasjon, så en bruker gjerne ReLU som aktiveringsfunksjon for å innføre ikke-lineæriatet. Ved å gjøre sub-samling (pooling) får en igjen redusert antall features (f.eks maxPooling 3x3 reduserer med en faktor på 9). Pooling "glatter" dataene og reduserer spatial oppløsning (gir bla translasjonsinvarians, dvs ikke avhengig eksakt posisjon for feature).

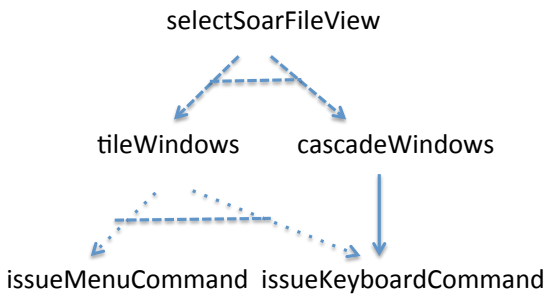
Oppgave 3 (20%)

a)

```
Selection rule set for the goal: selectSoarFileView                1
If task is use tile windows then accomplish goal: tileWindows              2
If task is  use cascade windows then accomplish goal: cascadeWindows
Return with goal accomplished                                      14


Selection rule set for the goal: tileWindows                       3
If task is tile windows using menu then accomplish goal: issueMenuCommand 4
If task is tile windows using keyboard then accomp.goal:issueKeyboardCommand
Return with goal accomplished                                      13
```

```
Method for goal: issueMenuCommand            5
Step 1. Locate Menu Bar                      6 - CP
Step 2. Move cursor to menu                  7 - P
Step 3. Press mouse button // menu appears   8 - B
Step 4. Move cursor to command name in menu  9 - P
Step 5. Verify inverse                       10 - M
Step 6. Release mouse button                 11 - B
Step 7. Return with goal accomplished        12


Method for goal: issueKeyboardCommand
Step 1. Press cmd-key   // key must be down until command letter typed
Step 2. Type command letter // letter is t, implicit WM-access
Step 3. Release cmd-key
Step 4. Return with goal accomplished

Method for goal: cascadeWindows
Step 1. Accomplish goal: issueMenuCommandFromMenuBar
Step 2. Return with goal accomplished
```

Målhierarkiet ser da slik ut:



Dotted line: selection rule

Et **dypt målhierarki** kan bety **lite gjenbruk** og **større kognitiv kompleksitet** siden det er mer å holde rede på (sub-mål) for å utføre en task.

b)

T = antall_step*100ms + Σ KLM-operatorer

= **adminstrasjonsledd kognitiv prosessor** på 100 ms pr. step utført + sum av utføring av **KLM-operatorene**

Oppgaven ber egentlig om beregning for alle alternativene. Her et eksempel på beregning (ett alternativ med step nr og KLM-operatorer satt inn i koden over) Viktig her å huske å ta med administrasjonsleddet. Dessuten å vise `trace` **ved å skrive linjenr utført bak koden i a), her 1 til 14, og evt. KLM-operator (som vist). Dette gir 14 step+ KLM-operatorene:**

T = 14*0.1 + (CP+2P+2B+M) = 1.4 + 1.2 + 2*1.1+ 2*0.1+ 1.2) = 6.2 s

c) **Kreative oppgaver** kan **ikke** modelleres, bare rutine-oppgaver.

Operatorene lar seg **parallellisere** siden de utføres i hvert sitt sub-system (motor, kognitivt eller perseptuelt) som kjører i parallell. **Eksempel:** Locate og Move i Step 1 og Step2 i metoden for `issueMenuCommand` over kan slås sammen til ett step.

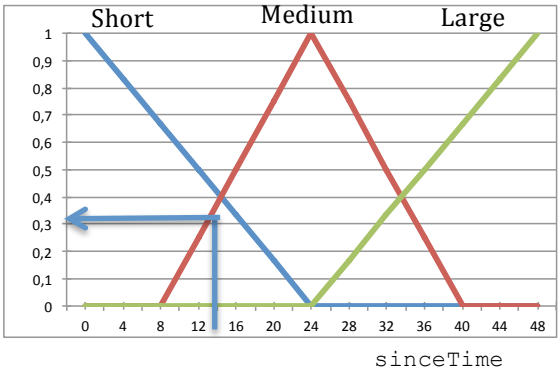
Tiden er da skal bruke er **den største** av KLM-operatorene.

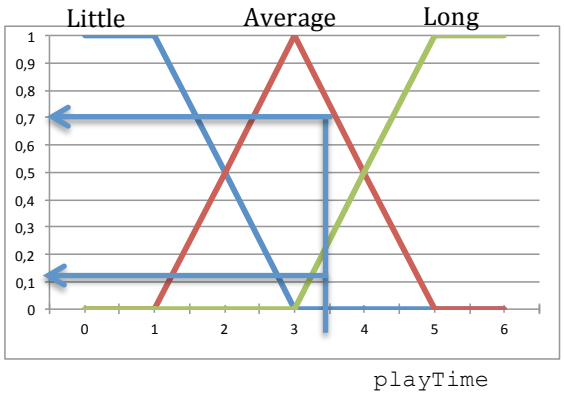
Kan evt. her også nevne (men ikke krav): Seleksjonsregler inneholder paralle-if-setninger hvor alle fyrer samtidig og bare en velges. Her er det regel-baserte produksjonssystemet (kognitivt sub-system alene) som utfører dette.

Oppgave 4 (20%)

Det viktige her er at man viser hvordan ting beregnes med de oppgitte reglene: Fuzzification, Evaluation, Aggregation og Defuzzification.

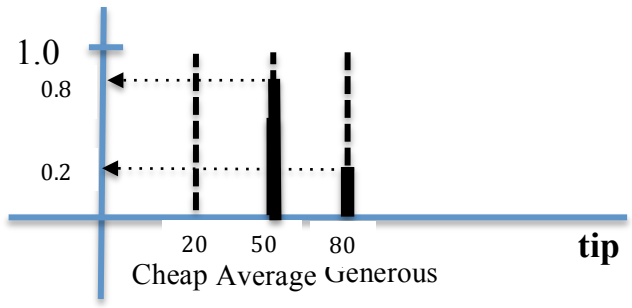
a) **Fuzzification** :Leser av settmedlemskapet for verdiene ut fra figurene gitt i oppgaven. Ser at det er Sugeno regler med 3 singeltons. Dette gir følgende **evaluering** (verdier er vist over regelene):






```
      0.0                              0.0
If (playTime is Little) then emotion is 20
      0.8              max   0.4                  0.8
If (playTime is Average OR sinceTime is Medium) then emotion is 50
      0.2              min    (1-0.4)              0.2
If (playTime is Long AND sinceTime is NOT Short) then emotion is 80
```

Overført verdi til høyre side av regel er gitt ved verdien vi får ved at venstre siden evalueres gjennom at OR = max(), AND = min() og NOT = 1-val.
I Sugeno beregnes **aggregering og defuzzification** som et vektet gjennomsnitt:



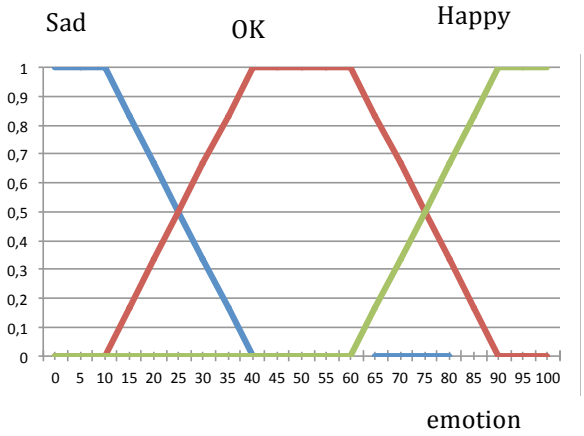
$$WA_{Sugeno} = \frac{\sum_{i=1}^{n} \mu_i x_i}{\sum_{i=1}^{n} \mu_i}$$

Singeltons med lengde 1.0 klippes til verdi gitt fra høyreside av reglene over.

WA = 0*20+50*0.8+80*0.2/ (0.0+0.8+0.2) = 56

Kommentar; Regel 3 burde ha brukt NOT Long og ikke NOT Short (se rød utheving i regelen), men det spiller ingen rolle for oppgaven…

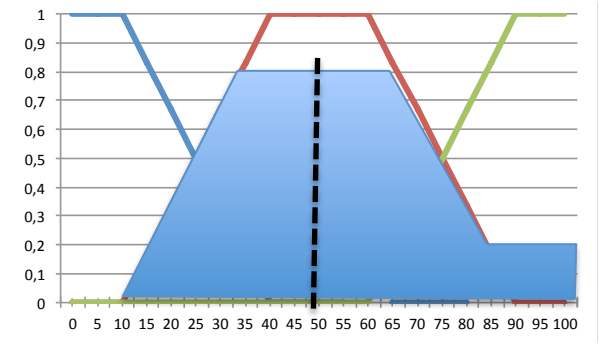
b) En må her selv "fritt" velge, samt tegne opp ut-sett for emotion (Mamdani). Beregningen bære preg av hva en velger av output-set og step-size. Her et mitt valg/tegning :



Reglene blir i Mamdani og gir samme evaluering (verdi til høyre side):

```
If (playTime is Little) then emotion is Sad
        0.8                  max   0.4                    0.8
If (playTime is Average OR sinceTime is Medium) then emotion is OK
        0.2                  min     (1-0.4)              0.2
If (playTime is Long AND sinceTime is NOT Short) then emotion is Happy
```

Klipper ut-settene med verdiene:



Gjør deretter aggregering av de tre klippede settene (blått areal):

$$COG = \frac{\int_a^b \mu_A(x)\, x\, dx}{\int_a^b \mu_A(x)\, dx}$$

GOG =Centre of Gravity

Beregner nå **COG** (vist som svart stiplet linje i figuren over). Velger step på 5 i integreringen:

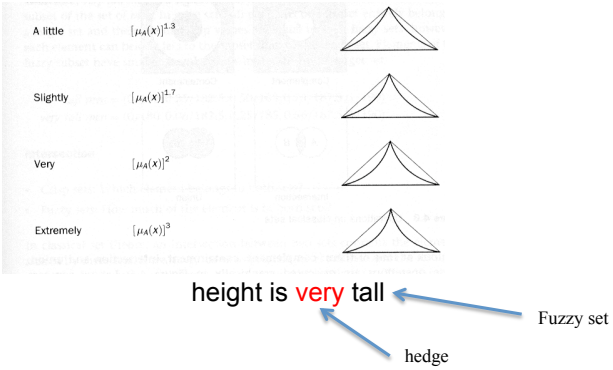
COG = (0*0+0*5+0*10 + 0.2*15+0.4*20+0.5*25
+0.7*30+0.8*(35+40+45+50+55+60+65)+0.6*70+0.5*75+0.3*80+0.2*(85+90+95+100)/(0+5+10+15+20+25+30+35+40+45+50+55+60+65+70+75+80+85+90+95+100)= ca 47

som gir et resultat litt forskjellig fra Sugeno pga valg av fuzzy ut-sett (verdie vil variere for hver

Hedger gir et rikere språk for å skrive regler. Hedgene (very, little, extremely etc..) endrer formen (membership) på aktuelt sett ved at hver hedge utfører en matematisk funksjon på μ.(kvadrering, rot, etc.).

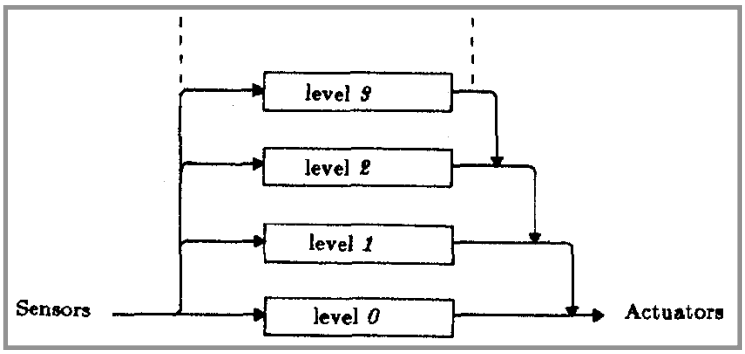
Eksempelregel:

```
If (distance is extremely small) then brake is Hard
```

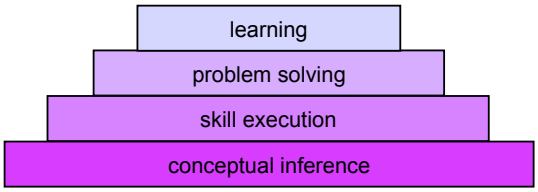


Oppgave 5 (20%)

a) **Brooks sub-sumption-arkitektur** har **ingen** intern modell, men bruker verden direkte (som ”modell”) i sin resonnering ved sensing. Hoved-ideen med arkitekturen kommer fra naturen,”Follow the evolutionary path of intelligence: Start with simple intelligence .After a successful design, extend to higher levels of Intelligence”
Dette kommer til syne gjennom en **lagdelt arkitektur**: det overliggende lag tar opp i seg (subsumes) laget under. På den måten er den mer robust og reaktiv samt parallell og distribuert. Kommunikasjon mellom moduler i lagene: Inhibit gjør at all output går tapt for en tid og supress erstatter input. I tillegg brukes altså verden direkte (ingen intern modell)



b) **Moduler** Icarus:



Laveste modul er **Conceptual inference** som lager beliefs (concept-instanser) ved at concept hierarkiet blir instansiert bottom-up med å starte med lav-nivå sensory inputs:

```
Repeat on every cycle (make up-to date beliefs):
- Update contents for perceptual buffer (low-level sensory data)
- Recognizes concepts:
      -primitive concepts at the lowest level by matching them
      against the perceptual elements (may match against more than
      one object or set of objects)
      -then match higher-level concepts against instances of the
concepts
```

En **impasse** oppstår når Icarus ikke kan finne en "applicable skill path". Icarus kan ikke finne et skill (feiler) for å løse et goal og må gå over til problemløsning. (Skill er en metode/prosedyre som kan være hierarkisk og som driver Icarus mot en måltilstand). Icarus gjør problemløsning slik: når condition delen i et skill feiler, prøver en å løse dette ved å se etter (primitive) **skills som matcher ikke-oppnådde mål i condition**-delen. Dersom en ikke finner et **skill**, prøver en å finne et **konsept** som ikke er oppfylt og så utføre andre skills for oppfylle konseptet for å komme videre (beliefs).

c) "The **Heuristic Search Hypothesis**" sier: "The solutions to problems are represented as symbol structures. A physical-symbol system exercises its intelligence in problem- solving by search, that is, by generating and progressively modifying symbol structures until it produces a solution structure"

Soar gjør nettopp dette med at **reglene** utfører søk i tilstandsrom ved å foreslå, evaluere, og utføre operatorer som **modifiserer** nåværende tilstand <s> til en ny tilstand mot målet ,og som etterhvert kan lede helt frem til **målet** (solution structure).

Regelen nedenfor sjekker hele tiden om finnes en mugge med 4 liter (type monitor, dvs desired/**goal state** monitor). Regelen er bygd opp "if condition then action" med syntax: sp{ navn-på-regel, condition-liste **-->** action-liste}

```
sp {water-jug*detect*goal*achieved      // navn regel (fritt valg)
         (state <s> ^name water-jug     // første condition MÅ refere til state
                    ^jug <j>)           // test på ulike attributer i tilstanden
         (<j> ^volume 5                 // må være muggen med plass til 4 liter
              ^contents 4)              // og har innhold 4 liter
       -->                              // then
         (write (crlf) |The problem has been solved.|)   //aksjon 1 skriv ut
         (halt)}                                         // aksjon 2 stopp agent
```

Arkitekturen gir at en må lage **to** regler: en som **foreslår** operatoren fill til Soar (regeltype: propose i elaboreringsfasen), og en regel som **utfører** operatoren **hvis Soar velger den** i decision-fasen (regeltype: apply):

```
sp {water-jug*propose*fill              // navn regel (fritt valg)
         (state <s> ^name water-jug     // finn state <s> ned navn water-jug
                    ^jug <j>)           // og at det er en mugge <j>
         (<j> ^empty > 0)               // som har ledig plass
         -->                            // THEN
         (<s> ^operator <o> +)          // legg til <s> forslag (+) på operator
         (<o> ^name fill                // med navn fill
              ^fill-jug <j>)}           // og "parameter"/attributt fill-jug lik <j>
```

Merk: det er setningen: (<s> ^operator <o> +) som forteller at dette er en propose-regel. + angir at operatoren er acceptable og kan eventuelt velges.

```
sp {water-jug*apply*fill
          (state <s> ^name water-jug    // navn regel (fritt valg)
                     ^operator <o>      // sjekk om Soar har valgt operator <o>
                     ^jug <j>)          // og at det er en mugge <j>
          (<o> ^name fill               // og at operatoren er fill
               ^fill-jug <j>)           // med mugge <j>
          (<j> ^volume <volume>         // som har volum <volume>
               ^contents <contents>)    // og innhold <contents>
          -->                           // THEN
          (<j> ^contents <volume>)        // sett nytt innhold nå lik volum
               ^contents <contents> -)}   //slett gammel Verdi (har endret tilstand)
```

Merk at det er setningen: ^operator <o> som forteller at dette er en apply-regel (Soar genererer/legger ut ^operator <o> til tilstand <s> når **Soar har valgt** operatoren for oss slik at **vi** kan utføre den vha regelen).

d) Finner **frekvensbåndene** fra EEG ved først å gjøre **windowing** på signalet (siden ikke-periodisk), og deretter gjøre **FFT** (Fast Fourier Transform). **Energien** i frekvensbåndene kan da beregnes. Deler hovedbåndene opp i subbånd. Kan så lage en **digital kode** hvor hvert bit representerer et av båndene. Et bit settes til 1 dersom båndet har **minst 4% av den totale energien** til båndene (ikke krevd her: 4 bit δ , 4 bit θ , 5 bit α og 9 bit β)
Men når vi ser på kodene vi får når vi analyserer EEG under ulike emosjoner, ser vi **tvetydigheter**.
Løsning: se mer detaljert på hvilke sub-bånd som har **høyest energi** for et hovedbånd (f.eks . $\delta 1$ eller $\delta 4$ i δ-båndet)