**M0 main file**

Read an ini file and set global parameters.

Initialize the word list, the knowledgebase, and the working store.

Populate mapping between concept relation type letters and numeric codes.

Call M1.

\*\*\* defaults \*\*\*

Default r: 50

Default p: 100

Default c: 100

\*\*\*

\*\*\* concept code mapping \*\*\*

W: 0 //word

S: 1 //similar

D: 2 //identical

C: 3 //class 1:member 2:class

F: 4 //feature 1:entity 2:feature

Q: 5 //quantifier 1:quantifier 2:entity

A: 6 //action 1:actor 2:action

I: 7 //impact 1:actor 2:impacted object

R: 8 //relative 1:speficier 2:entity e.g. with, hand

T: 9 //time 1:entity(action) 2:time-expression

P: 10 //part, possessing 1:owned entity/part 2:owner/whole

M: 11 //more, ordered list. 1:feature1 2:feature2 …

IM: 12 //implication 1:reason 2:consequence

N: 13 //necessary condition 1:condition 2:consequence

V: 14 //relevance

AND: 15 //and

OR: 16 //or

NOT: 17 //not

XOR: 18 //exclusive or

\*\*\*

**M1 Mentalese input**

Input: test file (like testfile.txt ) that has english and corresponding mentalese, includes also the questions and the expected answers to the questions.

process input (from file), later receive the output of mentalese translator.

Call readtest implemented in testing.py

This will populate a vector with mentalese expressions, including the questions.

For each mentalese row

* Call **Main loop of input processing.**

**Main loop of input processing:**

Input: mentalese sentence.

1. walk through all assumptions (not being dead)
   1. take next item „a” from input (completed concept or next word), continue the assumption
   2. if not word,
      1. if not question,
         1. M3: Store compound concept in WM
         2. perform reasoning
      2. if question,
         1. Store compound concept
         2. M3: basic question answering
         3. Delete question from WM
         4. Evaluate answers
   3. if word, generate all word meaning assumptions, walk through these
      1. M3: Store word in WM, the i-th meaning of „a”
      2. perform D-reasoning, C-reasoning, IM-reasoning
      3. generate all mapping assumptions (if any), walk through these
         1. M6: Map concept (store D-relation)
         2. perform D-reasoning, C-reasoning, IM-reasoning
         3. M6: update lm value of assumption
         4. Suspend this mapping assumption
      4. keep top mapping assumption, kill all others
   4. keep top word meaning assumption

**M2 Word list**

Add new word. Input: word.

* *Determine if it is a word. (comes later)*
* If word, add to word list and knowledgebase:
  + append word to word list. (fixed size)
  + Call M3 to add concept in the KB, type=W
  + Set pointer in the word to the new KB concept.

Find word. Input: word.

* Search word in word list.
* Return pointer or not found flag.

Get number of meanings. Input: pointer to word.

* Return number of meanings.

**M3 Concept**

Concept class

\*\*\* concept specification \*\*\*

* Fields
  + Concept relation type
  + r, relevance
  + p, probability
  + c, confidence
* link to KB, knowledgebase
  + pointer to the same concept in the knowledgebase
* link to WL, word list
  + index in word list, if this concept is a word
* parents
  + pointers to parents (in same list)
* children
  + pointers to children (in same list)
* future plan: to have some id for concepts. Currently, the index in the KB or WM list is used.

**Kbase** class

This has two instances:

KB, the knowledge base

WM, the working memory

\*\*\* specification \*\*\*

* list of concepts
* current index
* future: largest ever id used (to generate a new id)

Add concept. Input: store (knowledgebase/working memory), number of parents, pointers to parents, and concept type. (for word there are no parents) optionally link to WL, link to KB.

* Set parent pointers.
* Set concept relation type.
* Set child in the parents, if this is not a word.
* If this is a word (relation=W):
  + Parent is empty
  + Set link to WL
  + If this is in WM, set link to KB
* Return pointer to new concept (or the index of new concept)

Set r (input: pointer to concept, r)

Set p

Set c

Store word in WM : read\_concept

* Read\_concept gets the concept string like C(dog,animal), and stores both the words, and the compound concept.
* For word, it calls

Store compound concept in WM : read\_concept Input: concept strings, separated by space, as a single item in a list.

* + - M3: Create next empty concept in working store, with pointers to parents (use a mapping between concept type letters in input, and concept codes)
    - M3: set r, p, c to default values
    - For all parent concepts, set pointer to this new concept as child
    - If parents have link to KB, call M4: search concept in KB based on parents
      * If success, set KB link - that will show to the same concept in the KB.

basic question answering input: question (as concept object)

* Search WM for answer search\_inlist , this will match all earlier concepts in WM to the question. If we have reasoning already performed, then more complicated questions can be answered.
* Search KB in the future when we have some content deleted in WM and added in KB.

question answering using mapping

1. walk through all assumptions (not being dead)
   1. when searching for the answer to F(Joe,?) all D(x,Joe) and D(Joe,y) need to be considered, so the search need to be extended to F(x,?) and F(y,?).
2. keep best answer

Start assumption (or hypothesis) – this is to fork the WM when more than one possibility needs to be evaluated against each other. Like for word sense disambiguation and for mapping.

Input: new assumption or continuation; last concept in WM

Suspend assumption – to stop evaluation in this assumption branch.

**M4 Knowledgebase KB and working memory WM**

Store concepts in long term memory.

Create empty concept in knowledgebase. Input: number of parents, pointers to parents, and concept type. (number of parents=1 for a new word, type=0 for word.)

* Call M3: create empty concept, with store=knowledgebase.

Save knowledgebase (to some automated file name)

Search concept based on parents (search needs to be done in KB+WM. If the concept is found in WM first, then continue search in KB. If both found, KB needs to be returned.)

For a compound concept in WS, it tries to find it in KB, based on parents.

Input: concept in WS.

Walk through all children of one parent in KB

Match relation type and other parent`s pointer

If matching, the concept is found, return its pointer

Stop after first match (so now a compound concept should have 1 meaning only)

Return pointer in KB (WM) – **main thing is that we need to return the concept in KB or the first occurrence in WM (or we make use of the D() mapping relations, and no matter which WM occurrence is returned)**

Save working store (to some automated file name)

**M5 Testing**

Read test file readtest

Evaluate test file eval\_test

**M6. Mapping**

Map concept input: concept „a” in WM (a word like „it” or a concept like Q(f,g) (e.g. this fox))

1. Use the KB-link of „a” (if no link, try call map concept based on synonim, EXIT)
   1. if a=Q() then arrive at b=Q(f,%1) (e.g. Q(this,%1)
2. browse through children of „a”, search for a D(Q(this,%1),%2) with r>r0 (r relevance will show the need for mapping, r0 is a global threshold) (M is the mapping relation)
   1. if none, try call map concept based on synonim
   2. EXIT
3. Set next mapping assumption branch in WM (H1, H2, ....)
   1. H1 means no mapping, suspend H1
   2. if not H1, take next D(b,%1)
      1. take next N(D(b,1%), x), condition. The mapping-rule is the N, for example x=A(%1,%2) which means the subject, mentioned previously.
         1. search for x in WM, stop if Mapping rule stop exceeded (e.g. A( , ) ) Walk through all x found:
            1. pick the %1 concept from WM, let this be d
            2. set up next mapping assumption, Hn, generate D(a,d) , store - here we originally had D(b,%1) and concluded to D(a,d)
            3. Update r-value of this assumption using update r in assumption based on N
   3. end while

Update lm value of assumption – likelihood measure; we want to increase if knowledge in a specific mapping or word sense assumption matches any other knowledge in WM/KB. Input: assumption start. (only latest start needs to be taken.)

Something like this: for all concepts within the last assumption (this will now include reasoning results plus input content as well) we search through the earlier WM content of the same branch, and KB. If found, compare p values. The result of this comparison is to be added to the lm of the assumption.

**M7. decision rules (model parameters)**

r0 global threshold for M relation can return now 0, this is to decide whether a mapping rule should be used or not.

Mapping rule stop for now can return “stop” if the search exceeds the previous sentence backwards.

update r in assumption based on N for now returns r value of the N relation. This is the way the mapping rule influences the acceptance of the mapping hipothesis.

keep top mapping assumption for now returns the top 1 or 2 by adding r+lm. This is to decide which to kill, which to keep.

keep top word meaning assumption now returns the top 1 or 2 by p+lm. This is to decide which to kill, which to keep.

keep best answer now returns the answer in the assumption with top r+lm. This is to decide which answer to choose.

compare p values of same concept now returns MIN(p1,p2). This is to find out if an assumption matches earlier knowledge.

**M8. C-reasoning**

Set up C relation with rule-type concept if the WM input is X(a,b) and we have X(a,%1) or similar in the KB then generate C(X(a,b),X(a,%1)).

Reasoning based on C Input: concept in WM, b. There are 3 cases where class for b can be found: 1 rule-type concepts wit % placeholders 2 classic C(b,c) relations 3 classes of mapped concepts.

1. if b is of form Y(a,x) then search Y(%,x) and Y(a,%) as children of y or a, because these two concepts are classes for b. let this be now c=Y(%,x).
   1. If found, then for any X(c,d) child of c, generate X(b,d) in WM. Copy over p value from X(c,d).
2. use the KB-link of “b” which shows to KB.
   1. Browse through children to find C(b,c) ( b is in class c)
   2. If found, then for any X(c,d) child of c, generate X(b,d) in WM. Copy over p value from X(c,d).
3. If b is mapped, like M(x,b), walk through mapping partners x
   1. Browse through children to find C(x,c) ( x is in class c)
   2. If found, then for any X(c,d) child of c, generate X(b,d) in WM. Copy over p value from X(c,d).

(seems that there is no need to browse through all concepts in WM to see if any of these is C(b,c). If there was such a concept, it was among children of b, and b would be either stored in KB or an earlier input in WM. In both cases the new occurrence of b is linked to it. So we can start with using the KB-link. If b has multiple occurences, then they are mapped or not. If not, they are separate instances. If mapped, maybe C-reasoning needs to be performed again, this is in 2.)

**M9. IM-reasoning**

Store IM input in WM If the mentalese input is i=IM(a,b) then store i, a and b in WM with p=0.5.

**Example for rule:** IM(AND(D(%1,%2)p1,X(%1,%3)p2),X(%2,%3)p=d\_rule)

**RULE MATRIX**

d\_rule= [[0, 0, 0, 0, 0], [0, 1, 1, 1, 1], [0, 1, 1, 2, 2], [0, 1, 2, 2, 3], [0, 1, 2, 3, 4]]

p value of conclusion=d\_rule[p2][p1]

**rule\_reason**

IM-reasoning using %-rules

This will enable to implement C-reasoning and most other reasoning as simple mentalese input.

Input: last context in WM (probably a question)

Walk through **all** concepts in WM, let the next one be Z(a,b) (current concept):

1. Find out if there is a rule to apply for Z(a,b). Call **find\_rule**(index of current concept). It returns the list of [index of the terminal rule (IM() concept)]. The first concept of the first argument of IM() must match Z(a,b), this must be enforced.
2. Walk through rules to apply. It has the format IM(g,k).
   1. Take g or walk through arguments of g if it is AND relation. (in a general version, this is a recursive walk on the tree of arguments and their matches.) (see **condition** list: list of concepts in the first argument of the IM() relation.)
      1. First concept must match Z(a,b). Perform **rule\_collect**.
      2. Second concept: search WM for matches. (proper search needed)
         1. Walk through matches.
            1. Perform **rule\_collect**. Input: condition concept, matching concept.
            2. Perform **generate\_concept** using someRule.matchrelation, someRule.matchvalue, someRule.matchp, and the rule matrix.

**find\_rule**

Input is index of Z(a,b)

Try find rules in KB like Z(%1,%2), Z(%1,b), Z(a,%1), X(a,b), X(a,%1), X(%1,b).

Call a general kb\_search function, which searches KB starting from the words.

If something found, go down in all children to see if we arrive at some IM() relation.

If yes collect list of these IM concepts.

**rule\_collect**

Input: index of the condition concept, index of the matching input in WM

Here we found a concept that is matching the condition of a rule.

1. If the condition has relation=X (meaning anything), set **matchrelation**.
2. someRule.condition[i] should match someRule.match[i]
3. for this match ONLY, someRule.matchvalue stores the concepts corresponding to %1, %2, %3, ….
4. for this match ONLY, someRule.matchp stores the p values corresponding to p1, p2 , …

**generate\_concept**

Input: all we had from rule\_collect, and index of the rule IM() concept.

In k we have some %1, %2 … concepts, these need to be replaced based on rule\_collect info.

P needs to be set using the rule-matrix given in k.rule.

Store k in WM.

implication\_reasoning

Generate concepts based on IM relation

Input : IM-concept in WM, i=IM(a,b) ( a and b stored).

1. search „a” in WM, if found, copy p value to “a”.
2. Map „a” in WM (see mapping below) -> this may result a D(a,e) relation.
   1. if mapped, copy p value from e to a.
3. If not mapped,
   1. search „a” in KB, if found, copy p value from KB to „a”
4. take „b”
   1. let p1 be the p-value of i and p2 the p-value of „a”
   2. set p of b = MIN(p1,p2) or apply a rule-matrix to get p of “b”.
      1. use rule-matrix “pim”.

**Mapping example using D-relation**

*Joe and Mary are playing in the sandbox. He is her brother.*

KB tartalom:

IM(F(person,male), NOT(F(person,female)))

IM(F(person,female), NOT(F(person,male)))

Vagy valami mas kifejezese annak hogy ezek egymast kizarjak,

peldaul F(person, AND(male, female)) p=0 ezt tomoren kifejezheti

vagy AND(F(person,male),F(person,female))p=0.

C(Joe, person)

F(Joe, male)

C(Mary, person)

F(Mary, female)

C(he, person)

F(he, male)

C(brother, person)

F(brother, male)

D(he, %1) vagy az IM-reasoning segitsegevel: IM(he, D(he,%1))

N(D(he,%1), A(%1,%2)) - N a “condition”, feltetel relacio, D(he,%1) biztosan nem igaz ha A(%1,%2) nem igaz.

Ez a felhasznalando mapping rule.

Input alakulasa a WM-ben:

Joe F(Joe, female)p=0 and Mary F(Mary, male)p=0 AND(Joe,Mary) … A(Joe,play) A(Mary, play)… he **H11** D(he, Mary) C(he, person) F(he, female) F(Mary, male) **H11end** **H12** D(he, Joe) C(he, person) F(he, male) F(Joe, male) **H12end H13 H13end H12** is her brother Q(her, brother) D(he, Q(her, brother)) …

1. Joe utan C-reasoning es IM-reasoning eredmenye hogy F(Joe,female)p=0. Hasonloan F(Mary,male)p=0.
2. H11 helyen Map concept fut, “he” gyerekei kozott megtalalja hogy D(he,%1). Megtalalja D(he,%1) gyerekekent hogy N(D(he,%1), A(%1,%2)). Emiatt keresni kezd visszafele egy A() conceptet, megtalalja hogy A(Mary,play). Felveszi a H11 elagazast, es WM-be a mappinget D(he, Mary).
3. Most kovetkezik a reasoning (C, IM, D reasoning) amibol most D-reasoning mukodik, ebbol felveszi a WM-be hogy F(he, female)p=1 es F(Mary, male)p=1.
4. Most fut update lm value of assumption H11-re, osszehasonlitja F(Mary,male)p=1 a korabbi F(Mary,male)p=0 concepttel, ami negative lm update-et eredmenyez, mondjuk lm=-100, ezt a H11-hez rendelve tarolja.
5. H12-ben lm update pozitiv, +100, ezt H12-hoz rendelve tarolja.
6. H13 lm-je nulla.
7. Most fut keep top mapping assumption ami ezek kozul valaszt: H11 r=100 lm=-100 H12 r-100 lm=100 H13 r=0 lm=0 , kivalasztja H12-t, a masik kettot killed flaggel latja el.
8. Veszi a kovetkezo szot, is, mar csak a H12 agban.

“fox” mapping example

This example shows that we can force a mapping assumption by explicit translation rule.

e/ The fox is one of our few mammal predators. Wolves and Mountain Lions have long been extinct in Virginia.

m/ C(fox, Q(one of,Q(our,F(predator,few,mammal)))) F(A(AND(wolve,Mountain Lion),extinct),long,R(in,virginia))

mapping/ D(we,P(people,virginia))

mapping/ C(AND(wolve,Mountain Lion), F(predator,few,mammal,R(in,virginia)))

To make this work we make use of an abstract concept of recently mentioned group of people.

Grammar parser: our NP mentalese: Q(our,NP) we

This will make “we” appear in WM.

KB content: IM(C(recent group of people,%1),D(we,%1))

Mapping, which is the D(we,%1), will be used if we have something as recent group of people.

For the second sentence we need this KB content:

We have the abstract concept of recently mentioned actor, and recently mentioned class. These are set up in KB:

Sentence: C(fox, Q(one of,Q(our,F(predator,few,mammal))))

KB used: IM(C(%1,%2), D(%1, recent actor), D(%2, recent class))

“… predators” become identical to recent class.

Second sentence: F(A(AND(wolve,Mountain Lion),extinct),long,R(in,virginia)) or similar

KB used:

1. IM(A(%1,%),D(%1,recent actor))
2. IM(F(%1,%), D(%1,recent actor))
3. IM(I(%,%1),D(%1,recent object))
4. IM(C(%1,%),D(%1,recent actor))

Mapping rules in KB:

1. IM(AND(D(%1,recent actor),D(%2,recent actor)),D(%1,%2)) // two recent actors may be mapped on each other
2. IM(AND(D(%1,recent actor),D(%2,recent class)),C(%1,%2)) // a recent actor may be mapped to be member of a recent class
3. IM(AND(D(%1,recent actor),D(%2,recent actor)),OR(C(%1,%2),C(%2,%1))) // two recent actors may be mapped into class-member relation

Finally,

“Wolves and Mountain Lions” are assumed to be member of “ …predators”.

The point here is that we DO HAVE something in *recent class.*

**How mapping gets confirmed in this example:**

1. **Wolves and mountain lions get mapped to something in neighbouring sentences: r +100**
2. **Wolves and mountain lions are mammals: lm +100**
3. **Wolves and mountain lions are predators: lm +100**
4. **If something is extinct then it`s class is few: lm +100**

The example above does not need to mention abstract concepts in the mentalese translation module. But we may want to make an exception and allow mention mapping related concepts there. Then we could have “map recent actor”, “map recent class” etc etc abstract concepts in the translation module.