**Obraz świata**

Agent znajduje się w świecie. Świat jest ograniczony tylko do obiektów ,które posiadają cechy. Cechy opisują obiekty, cechami mogą być na przykład kolory. Obiekt może posiadać wiele cech, ale poszczególne typy obiektów mają ściśle określony zestaw cech. Typy obiektów tworzą hierarchię ,każdy typ obiektu może mieć zbiór typów obiektów podrzędnych i nadrzędnych . Każdy typ obiektu może należeć do klastra będącego pewnego rodzaju abstrakcyjnym wyodrębnieniem typów obiektów posiadających cechy wspólne. Poszczególne instancje typów obiektów nazywamy modelami indywiduowymi. Modele indywiduowe mają swoje identyfikatory. Agent istnieje w świecie i obserwuje go. Obserwacja dotyczy tylko jednego modelu indywiduowego. Obserwacje podejmowane są w epizodzie, epizod to moment w czasie ,w którym agent dokonuje jednego ciągłego procesu percepcji. Obserwacja to obiekt model indywiduowy, zaobserwowane cechy, i ich stan. Cecha obiektu występuje w trzech stanach – {Jest, nie jest, być może}. Obserwacje z jednego epizodu zapisywane są w profilu bazowym, profil bazowy zawiera …

**Sensor Buffer Memory:**

1. Observation.

Obserwacja składa się z Identyfikatora, listy krotek (Cecha, Stan) oraz epizodu wyrażonego integerem, choć biorąc pod uwagę naturę języka pythona, również epizody wyrażone stringami lub innymi prostymi typami będą brane pod uwagę, epizody wykorzystywane są wyłącznie do porównywania, więc w zasadzie każdy obiekt ze zdefiniowaną metodą \_\_eq\_\_ byłby odpowiedni. W jednym epizodzie możemy dokonać wielu obserwacji, włączając w to dwie obserwacje tego samego obiektu reprezentowanego identyfikatorem (Dwie obserwacje w jednym epizodzie mogą wiązac się z szybką zmianą stanu). Obserwacje są atomowe, tj. dotyczą jednego obiektu reprezentowanego identyfikatorem. Obserwacje są sobie równe wtedy i tylko wtedy gdy mamy do czynienia z równymi identyfikatorami ,listą krotek oraz epizodem Obserwacje są potencjalnie równe kiedy mają te same identyfikatory oraz tą samą listę krotek, nie sprawdzając jednak epizodu.

**Long Term Memory:**

1. Woke Memory

Pamięc świadoma agenta. Posiada kolekcję profilów bazowych, modeli indywiduowych, klastrów oraz holonów. Potrafi zwracać holony przetrzymywane w pamięci, jeśli nie jest w stanie znaleźć pożądanego holona, tworzy go na życzenie klienta. Pamięc posiada pole point\_of\_no\_return ,który jest minimalnym epizodem do którego pamięc sięga tworząc holony. Przy zwracaniu/tworzeniu holonów brany pod uwagę jest czas podanej formuły, który wpływa na ostateczny kształt holona.

Semantyka, Język, komponenty.

1. State

Stan to enumerator reprezentujący zbiór {IS,IS\_NOT,MAYHAPS}. Enumerator posiada zdefiniowane operacja negacji, alternatywy i koniunkcji.

Negacja

|  |  |
| --- | --- |
| Q | Not Q |
| IS | IS\_NOT |
| IS\_NOT | IS |
| MAYHAPS | MAYHAPS |

Alternatywa

|  |  |  |
| --- | --- | --- |
| P | Q | P v Q |
| IS | IS | IS |
| IS | IS\_NOT | IS |
| IS | MAYHAPS | IS |
| IS\_NOT | IS | IS |
| IS\_NOT | IS\_NOT | IS\_NOT |
| IS\_NOT | MAYHAPS | IS\_NOT |
| MAYHAPS | IS | IS |
| MAYHAPS | IS\_NOT | IS\_NOT |
| MAYHAPS | MAYHAPS | MAYHAPS |

Koniunkcja

|  |  |  |
| --- | --- | --- |
| P | Q | P v Q |
| IS | IS | IS |
| IS | IS\_NOT | IS\_NOT |
| IS | MAYHAPS | MAYHAPS |
| IS\_NOT | IS | IS\_NOT |
| IS\_NOT | IS\_NOT | IS\_NOT |
| IS\_NOT | MAYHAPS | MAYHAPS |
| MAYHAPS | IS | MAYHAPS |
| MAYHAPS | IS\_NOT | MAYHAPS |
| MAYHAPS | MAYHAPS | MAYHAPS |

Enumerator wykorzystywany jest po to aby oznaczyć stan zaobserwowanej cechy danego obiektu. Operacje na nich mogą być potrzebne gdy w jednym epizodzie dojdzie do dwóch obserwacji jednego obiektu.

1. Tense

Enumerator reprezentujący zbiór {Past, Present, Future}. Jest brany pod uwagę przy budowaniu holonów, w przypadku czasu przeszłego bierzemy pod uwagę wszystkie obserwacje przed danym epizodem, w przypadku czasu teraźniejszego budujemy holon bazujący tylko i wyłącznie na ostatnim epizodzie. Implementacja czasu przyszłego nie istnieje. Czas jest argumentem opcjonalnym przy budowie formuł.

1. Trait

Cecha reprezentuje właściwośc obiektu, w tym momencie jest to reprezentacja słowna, ale w przyszłości cecha powinna zostać wzbogacona o formy wyrazu. Obiekt posiada kilka cech, predefiniowanych przez typ obiektu, innymi słowy dany obiekt nie może mieć cech które nie przynależą do jego Object Type’u

1. Modal Operator

Enumerator reprezentujący zbiór {Bel, Pos, Know, NoIdea}, zbiór reprezentuje kolejne obszary pewności na jakiś temat. W naszym przypadku jest to najczęściej poziom pewności na temat posiadania jakiejś cechy przez obiekt. Progi modalne zawarte są w pliku csv, dla wygody zmiany.

1. Logical Operator

Kolejny Enumerator reprezentujący zbiór {OR, AND ,XOR}, symbole wskazują na wartości logiczne formuł i decydują o tym jak przebiegnie wnioskowanie.

1. Formula

Interfejs odpowiedzialny jest za dyktowanie metod do implementacji klasom pochodnym. Formuła jest odzwierciedleniem zdania znanego z języka naturalnego, w zależności od formuły jednak ,składowe różnią się. Formuły służą do późniejszej budowy elementów takich jak Interrogative (Pytanie) lub Declarative(Zdanie oznajmujące)

1. Type Of Formula

Enumerator potrzebny do rozróżnienia poszczególnych rodzajów formuł, obecnie reprezentuje zbiór {‘Simple Formula’, ‘Complex Formula’, ‘Object Type Formula’}

1. Simple Formula

Klasa implementująca metody ,których sygnatura opisana jest w klasie Formula, Simple Formula odzwierciedla zdanie proste i pobiera następujące argumenty – Model indywiduowy, Cecha, Stan, Czas (Opc).

1. Complex Formula

Klasa implementująca metody ,których sygnatura opisana jest w klasie Formula, CF odzwierciedla zdanie złożone i pobiera następujące argumenty , Model indywiduowy, listę cech, listę stanów, operator logiczny oraz opcjonalnie czas.

1. Complex Formula OT

Klasa implementująca metody ,których sygnatura opisana jest w klasie Formula, CFOT odzwierciedla zdanie złożone w którym przedmiotem zainteresowania nie są cechy modelu indywiduowego, ale modele indywiduowe. Możemy np. wykreowac pytanie ‘Czy był dziś u Ciebie Kamil i Ambroży?’.

Semantyka, Język, konstrukty.

1. Sentence

Sentencja, interfejs służący do ustalenia sygnatury poszczególnych rodzajów sentencji. Sentencja korzysta z danych zawartych w formule, tych samych danych można użyć zarówno jako pytania, jak i zdania oznajmującego.

1. Sentence Type

Typ sentencji będący enumeratorem który reprezentuje zbiór {Imperative, Declarative, Interrogative, Exclaimatory} w tym momencie wykorzystywane są Declarativy tj. zdania oznajmujące i Interrogativy, tj. pytania. Dystynkcja jest konieczna jako, że różne sentencje są różnie traktowane.

1. Declarative

Klasa implementująca interfejs Sentence. Declarative jest odpowiedzialna za budowę zdań oznajmujących dla każdego rodzaju formuł. Obiekt zbudowany przez Declarative jest łatwo przekształcalny do postaci stringa, która to może być przeczytana przez moduł głosowy. Declarative jako argumenty przyjmuje subject, traits, states, logicaloperator, modaloperator, tense, każdy z tych argumentów jest opcjonalny.

1. Interrogative

Z polskiego zdanie pytające, ma za zadanie przekształcenie danych do postaci formuły (Możliwe dla każdego rodzaju formuły), klasa zawiera kilka metod o których warto wspomnieć . - - build\_from\_scraps to metoda ,która jako argument przyjmuje string i przekształca go w obiekt klasy Interrogative zawierający pełnoprawną formułę.

Get\_epistemic\_conslusion ,która pozyskuje odpowiedni holon i na jego podstawie wybiera pożądany operator modalny.

Check\_epistemic\_scope metoda która dla danej tablicy wartości zwraca tablicę odpowiadających im operatorów modalnych, granice wartości operatorów modalnych są zdefiniowane w pliku XML, dla wygody użycia.

Interrogative przyjmuje następujące argumenty subject, traits, states, logicaloperator, plaintext, memory, episode, tense z których każdy jest opcjonalny.

Semantyka, probiotyki wiedzy.

1. Config.xml

Plik konfiguracyjny zapisany w xmlu ,zawierający wszystkie zmienne systemowe, które są wykorzystywane w późniejszych procesach , np. granice dla poszczególnych operatorów modalnych.

1. CSVReader

Klasa zawierająca metody ,które czytają pliki typu csv i odpowiednio je obiektują.

1. IDs.xml

Plik konfiguracyjny typu xml zawierający identyfikatory dla obiektów świata rzeczywistego.

1. ObjectTypes.xml

Plik konfiguracyjny typu xml zawierający id oraz dopuszczalne cechy object typów, na podstawie tych danych tworzone są później Object Type’sy

1. Observation.csv

Plik zawierający obserwacje zapisane w csv. Idealny do symulacji obserwacji.

1. XMLReader

Klasa zawierająca metody niezbędne do wydobycia danych z plików XML.

Semantyka, identyfikatory

1. Identifier

Interfejs dla wszelkiego rodzaju identyfikatorów, identyfikatory jednoznacznie identyfikuje obiekt w świecie.

1. QRCode

Klasa odwzorowująca przykładowy identyfikator obecny w świecie, QRCode implementuje interfejs Identifier.

1. UniqueName

Klasa odwzorowująca przykładowy identyfikator obecny w świecie, Unikalne imię implementuje interfejs Identifier.

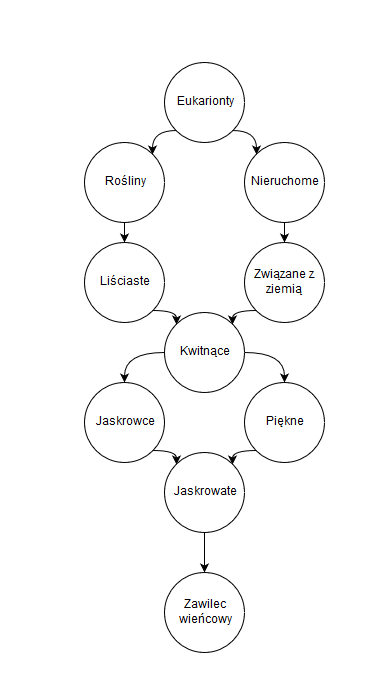
Semantyka, Meta kogniwistyka identyfikująca

1. Cluster

Klasa reprezentująca pewien stopień generalizacji wspólnoty jakiegoś zbioru obiektów. Np. każdy płaz należy do klastra zwanego królestwem zwierząt. Klastry używane są do ustalania hierarchii typów obiektów.

1. Object Type

Reprezentuje typ obiektu, Typem obiektu jest Dąb, jego szczególnym przypadkiem, czyli modelem indywiduowym jest Bartek. Typ obiektu opisuje cechy jakie może przyjąć model indywiduowy będący instancją tego typu obiektu. Obecnie OT przyjmuje 5 argumentów. Id, traits, superhylium , infraphylium, cluster. Trzy ostatnie są opcjonalne, klaster jest objaśniony powyżej, superhylium to typy obiektów nadrzędne do danego OT zaś infraphylium to obiekty podrzędne do tegoż OT.

1. Individual Model

Instancja object type’u Bartek typu Dąb. Jeden typ obiekty może mieć wiele modeli indywiduowych , zaś IM może mie tylko jeden object type. IM posiada unikalny identyfikator. IM przyjmuje jako argumenty identyfikator oraz typ obiektu.

Holony

1. Holon

Byt metafizyczny odzwierciedlający zdolnośc człowieka do budowania przekonań, umacniania i zmiany ich pod wpływem danych z zewnętrznego świata. Holon zawiera w sobie tao, będące proporcjonalnym odwzorowaniem zdania na pewien temat, zawierające informacje zarówno o wierze jak i braku wiary w dane zjawisko. Holony aby dbac o prawdę powinny być często aktualizowane.

Klasa holon jest interfejsem niezbędnym do budowy bardziej szczegółowych holonów.

1. Binary Holon

Holon binarny ,czyli taki który zajmuje się zdaniami prostymi, tao ma jedynie dwie połówki.

1. Non Binary Holon

Holon obsługujący zdania złożone, zarówno normalne jak i te poświęcone Objec Type’om.

Tao ma przyjmuje 4 wartości dla kolejnym formuł komplementarnych.

1. Grounder

Klasa Grounder służy do gruntowania przekonań bazując na Distributed Knowledge lub kontekście. Dokonuje się walidacji formuły na podstawie DK/Kontekstu.

Holony, Kontekst

1. UberContecst

Interfejs który implementowac powinien każdy Kontekst. Ustala sygnaturę metod obowiązkowych kontekstu.

1. Composite Context

Klasa implementująca UberContecst . Ustala kontekst, czyli zbiór profili bazowych które można rozważac jako podobne na podstawie funkcji podobieństwa. Przyjmuje argumenty: estimator – opisany niżej, bpset – zbiór Profili bazowych z których wyciągniemy kontekst, maxtreshold – wartość liczbowa od której zależy czy profil bazowy będzie zaliczony do kontekstu(im większy tym większy nacisk selekcyjny), taken\_into\_consideration – liczba ostatnich profili bazowych jakie będą brane pod uwagę przy tworzeniu grupy kontrolnej BP.

Holony, Kontekst , Estymatory

1. Estimate Functions

Interfejs służący do określenia sygnatury estymatorów.

1. Estimator Kind

Enumerator złożony z jednego elementu {DistanceFunction} służący do rozróżniania estymatorów, w przyszłości.

Holony, Kontekst, Estymatory

1. Distance Estimator

Estymator implementujący EstimateFunctions ,który estymuje odległośc profili bazowych na podstawie ilości cech wspólnych.

**Short Term Memory:**

Episodic

1. BaseProfile

Klasa reprezentująca profil bazowy, zbiór obserwacji zawartych w jednym epizodzie.

1. Distrubuted Knowledge

Klasa reprezentująca uporządkowany zbiór wiedzy, w tym wypadku oparty na profilach bazowych oraz epizodzie.

**Cycle**

1. Preparations

Klasa odpowiedzialna za przygotowania do działania, ekwiwalent budzenia się. Ustawiane są typy obiektów,modele indywiduowe, na potrzebę symulacji ekstrahowane są obserwacje.

1. Rise and Shine

Klasa z adnotacją todo, wspierająca pełną obsługę głosową.

1. Single Thread Cycle

Symulacja z jednym mainem pokazująca przejście przez funkcje programu, reaguje na obserwacje, odpowiada na pytania.

1. Multi Thread Cycle

Symulacja wykorzystująca wielowątkowośc , klasa wykorzystuje cztery wątki – Słuchacza, odpowiadacza, obserwanta (paczacza) i umysł, każdy z wątków ma swój zakres obowiązków.

**Speech**

1. Phonograph

Klasa wykorzystująca moduł pyttsx3 służący do przekształcania stringów na dźwięk.

Zawiera metodę która zamienia string na dźwięk.

1. Voice Recognizer Google

Klasa wykorzystująca moduł speech\_recognition ,który pozwala na rozpoznawanie mowy , metoda wykorzystuje API googla, do skutecznego odczytania mowy wymaga połączenia z internetem i może zostać wykorzystany nie więcej niż 50 razy dziennie.

1. Voice Recognizer Sphynx

Klasa wykorzystująca mniej wyszukaną bibliotekę Sphinx,do poprawnego działania wymaga zdefiniowania słownika, w innym wypadku często wypluwa bzdury.

**Pojedynczy przebieg życia programu**

Przykład wielowątkowy.

1. Preparacje
2. Na potrzebę translacji utworzony jest słownik stanów
3. Lista typów obiektów ekstrahowana jest z pliku konfiguracyjnego
4. Lista cech zostaje sporządzona na podstawie listy typów obiektów
5. Lista identyfikatorów stworzona jest na podstawie pliku konfiguracyjnego
6. Lista modeli indywiduowych jest tworzona na podstawie identyfikatorów i OT
7. Tworzony jest Fonograf
8. Na potrzeby symulacji powstaje lista obserwacji z pliku konfiguracyjnego.
9. Utworzenie pamięci świadomej przy pomocy przygotowanych modeli indywiduowych
10. Tworzone są kolejno 4 wątki – 0 - słuchający, 1 - odpowiadający, 2 - umysłowy, 3 - przechwytujący (Obserwujący).
11. Utworzona zostaje tablica obserwacji (Kolejka)
12. Utworzona zostaje tablica aktywnych pytań (Kolejka)
13. Utworzony zostaje episoder zwracający obecny epizod.
14. Utworzona zostaje flaga nowych obserwacji
15. Utworzona zostaje tablica odpowiedzi (Kolejka)
16. Utworzony zostaje semafor w postaci tablicy z czterema wartościami, dla każdego wątku.
17. Praca konstruktora zakończona.
18. Main. Każdy z wątków jest rozpoczęty. Semafor aktywuje wątek 3 i 1.

Przebieg pracy wątku 3 – przechwytującego:

1. Sprawdź semafor, jeśli nie jest ustawiony na 1, zaśnij.
2. Jeśli semafor ustawiony na 1 to przechwyc obserwacje (W tym momencie z pliku config)
3. Ustaw flagę nowych obserwacji na 1, ustaw wartość semafora 2 na 1

Przebieg pracy wątku 2 – umysłowego:

1. Sprawdź semafor, jeśli nie jest ustawiony na 1, zaśnij.
2. Jeśli flaga obserwacji równa 1 to na postawie obserwacji i epizodu przygotuj profile bazowe a następnie je dodaj do pamięci.
3. Ustaw flagę obserwacji na 0
4. Ustaw swój semafor na 0
5. Sprawdź czy numer porządkowy epizodu modulo 5 równe jest 0
6. Jeśli tak, update’nij holony.
7. Jeśli nie ma aktywnych pytań, ustaw swój semafor na 0 (Dla pewności)
8. Jeśli istnieją aktywne pytania, wygeneruj odpowiedź i dodaj do tablicy odpowiedzi.
9. Zdeaktywuj swój semafor, aktywuj semafor 1 – odpowiadający.
10. Jeśli flaga nie jest aktywna, epizod skacze o jeden w przód.

Przebieg pracy wątku 1 – odpowiadającego:

1. Sprawdź semafor, jeśli nie jest ustawiony na 1, zaśnij.
2. Jeśli nie ma aktywnych odpowiedzi, ustaw semafor na 0 i zaśnij
3. W przeciwnym wypadku, wypowiedz odpowiedź i usuń ją z kolejki, powtórz dla każdego z elementów.
4. Ustaw swój semafor na 0

Przebieg pracy wątku 0 – nasłuchującego:

1. Sprawdź semafor, jeśli nie jest ustawiony na 1, zaśnij.
2. W przeciwnym wypadku sprawdź czy nadeszły pytania.
3. Jeśli zostały wypowiedziane, nadaj im epizod, wypisz je i ustaw semafor 2 na 1