Mateusz Woźniak 30.05.2023

Sztuczna inteligencja i inżynieria wiedzy – lista 3

1. Programowanie logiczne to paradygmat programowania, który opiera się na logice matematycznej i wnioskowaniu formalnym. W programowaniu logicznym programy są tworzone w formie zbioru faktów i reguł logicznych, a obliczenia polegają na przeprowadzaniu wnioskowania na podstawie tych faktów i reguł.
   Prolog jest językiem programowania logicznego, który wykonuje wnioskowanie, sprawdzając czy zapytania są spełnione na podstawie dostępnych faktów i reguł, oraz korzystając z mechanizmu rezolucji, który pozwala na wnioskowanie poprzez odnajdywanie dopasowań i rozwiązywanie konfliktów w logice.
   Prolog znalazł zastosowanie w dziedzinach takich jak sztuczna inteligencja, eksperci systemy, przetwarzanie języka naturalnego, bazy danych, robotyka i wiele innych. Jego deklaratywny charakter oraz zdolność do wnioskowania na podstawie logiki predykatów czynią go przydatnym narzędziem do reprezentacji wiedzy i rozwiązywania problemów opartych na relacjach logicznych.

2. Definicje pojęć

```
/* Definicja pojęć istotnych dla opisu mikrofalówki */
component(magnetron).
component(control_panel).
component(door_switch).
component(plug).

state(magnetron, ok).
state(magnetron, not).

state(control_panel, ok).
state(control_panel, not).

state(door_switch, ok).
state(door_switch, not).

state(plug, ok).
state(plug, not).

/* Definicja możliwych problemów */
problem(not_starting).
problem(low_heat).
problem(sparks).
problem(door_not_opening).
```

3. Interakcja z użytkownikiem

```
/* Wyswietlanie rozwiązań */
print_all([]).
print_all([Head|Tail]) :-
    writeln(Head),
    print_all(Tail).

get_first([Head|_], Head).
```

print_all pozwala na rekurencyjne wypisywanie elementów listy.

get_first pozwala na uzyskanie pierwszego elementu listy.

4. Definicja przykładowych problemów

```
/* Rozwiązania problemów */

possible_cause(A, state(magnetron, not),
               _, _, state(plug, ok),
            'Magnetron might be terminally damaged.') :-
               A \= door_not_opening.

possible_cause(B, _, _, _,
               state(plug, not),
    'Please, try plugging in the device') :-
    B \= door_not_opening, B \= sparks.

possible_cause(not_starting, state(magnetron, ok),
              state(control_panel, not), _, state(plug, ok),
               'Try pressing the buttons harder.').

possible_cause(low_heat, _, _,
               state(door_switch, not), state(plug, ok),
               'Please, make sure the door is closed and does not leak any heat').

possible_cause(sparks, _, state(control_panel, not),
               _, state(plug, ok),
    'Control panel needs changing').

possible_cause(sparks, _, _, _, state(plug, ok),
    'Please, unplug the device').

possible_cause(sparks, _, _, _, state(plug, not),
    'Please, USE THE FIRE EXTINGUISHER QUICKLY').

possible_cause(door_not_opening, _, _,  state(door_switch, not), _,
    'Door switch needs fixing').
```

Niektóre z problemów nie wymagają podania każdego z pól predykatu (wtedy w ich miejscu widoczna jest podłoga _), inne mają ograniczony zakres (B \= sparks oznacza, że zmienna B nie może być równa sparks), zaś pozostałe są ściśle określone. Należy zauważyć, że w języku Prolog zmienne rozpoczynają się od wielkich liter.

5. Predykaty wyszukujące rozwiązań problemów:

```
/* Rozwiązywanie problemów */

assign_problem(1, not_starting).
assign_problem(2, low_heat).
assign_problem(3, sparks).
assign_problem(4, door_not_opening).

assign_magnetron('no', state(magnetron, ok)).
assign_magnetron('yes', state(magnetron, not)).

assign_control_panel('no', state(control_panel, ok)).
assign_control_panel('yes', state(control_panel, not)).

assign_door_switch('yes', state(door_switch, ok)).
assign_door_switch('no', state(door_switch, not)).

assign_plug('yes', state(plug, ok)).
assign_plug('no', state(plug, not)).
```

Na początek zdefiniowałem kilka reguł, dzięki którym można było odkodować tekst uzyskany od użytkownika.

```
print_detailed(Problem) :-
    write('Was magnetron heavily used? (yes/no) '),
    read(Magnetron),
    write('Was control panel heavily used? (yes/no) '),
    read(CP),
    write('Is door switch working well? (yes/no) '),
    read(DS),
    write('Is microvawe plugged in? (yes/no) '),
    read(Plug),
    assign_magnetron(Magnetron, M),
    assign_control_panel(CP, C),
    assign_door_switch(DS, D),
    assign_plug(Plug, P),
    findall(Solution,possible_cause(Problem, M, C, D, P ,Solution),List),
    length(List, Size), writeln(Size),
    (   Size\=0 ->  print_all(List); writeln('Please, contact our customer support.')).

troubleshoot :-
    write('What is your problem? \n1 - Not starting\n 2 - Low heat\n 3 - Sparks\n 4 - Door not opening\n'),
    read(X),
    assign_problem(X, Problem),
    findall(Solution,possible_cause(Problem,_,_,_,_,Solution),List),
    length(List, Size),
    (   Size=1 ->  print_all(List); print_detailed(get_first(List))).

search(Problem) :-
    writeln('What may be happening/ what can you do:'),
    findall(Solution,possible_cause(Problem,_,_,_,_,Solution),List),
    print_all(List).
```

Pozostałe predykaty pozwalają uzyskać dostępne rozwiązania problemów.

troubleshoot oraz print_detailed odpowiadają uzyskanie dokładnych informacji o problemie, bez konieczności posługiwania się nazwami reguł.

W przypadku search należy podać problem, a uzyskamy wszystkie możliwe konfiguracje stanu komponentów i ich rozwiązania.