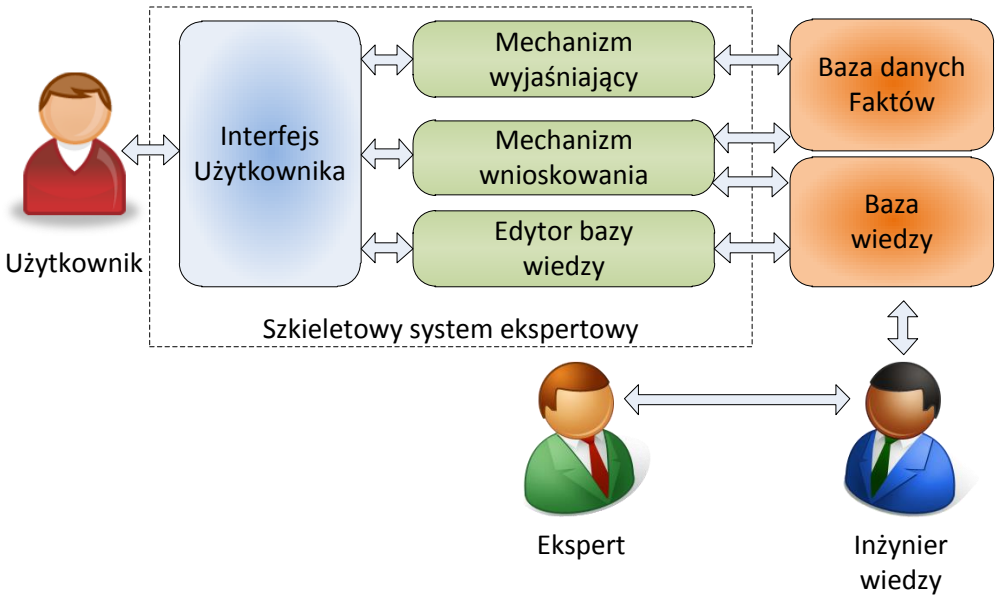
Wstęp

Systemy ekspertowe [3] to jedno z narzędzi informatycznych, którego celem jest pomoc podczas podejmowania decyzji. Typowe systemy ekspertowe można podzielić na:

- Systemy doradcze – których celem jest doradzanie człowiekowi podczas podejmowania decyzji. Przykładem takich systemów są systemy bankowe, gdzie program komputerowy doradza pracownikowi na temat udzielenia/nie udzielenia kredytu. Podobnie w medycynie, gdzie odpowiedni system służy do pomocy lekarzowi podczas przeprowadzania diagnostyki.
- Systemy podejmujące decyzje bez kontroli człowieka - systemy autonomiczne, w których maszyna/program podejmuje ostateczną decyzję bez udziału człowieka – np. sterowanie robotów przemysłowych
- Systemy krytykujące – których celem jest wskazanie sposobu osiągnięcia określonego wyniku, czyli znając problem, jak i jego rozwiązanie (wynik) system pokazuje jakie rozwiązanie osiągnięto (metodologię rozwiązania)

Cechą charakterystyczną wszystkich systemów ekspertowych jest to, iż zawierają one wbudowaną wiedzę niezbędną do podejmowania decyzji. Powoduje to, że przy tworzeniu systemu ekspertowego jego sposób działania zapisuje się w postaci deklaratywnej, czyli w odróżnieniu od tradycyjnych programów komputerowych, które tworzy się w postaci imperatywnej – ciągu instrukcji, które komputer/maszyna powinna wykonać, w programowaniu deklaratywnym określa się warunki jakie powinno spełniać rozwiązania – czyli co chcemy osiągnąć, nie precyzując jednoznacznie jak ten cel uzyskać.

Cała istota systemów ekspertowych zawiera się w dwóch głównych elementach ich budowy – bazie wiedzy oraz mechanizmie wnioskowania. Pełną strukturę systemu ekspertowego przedstawia Rys. 1



Rys. 1 Budowa systemu ekspertowego

Baza wiedzy zwykle reprezentowana jest w postaci reguł, natomiast **mechanizm wnioskowania** to algorytm określający sposób korzystania ze zgromadzonych informacji w **bazie wiedzy** i **bazie danych** (bazie faktów).

Tworzenie systemu ekspertowego sprowadza się więc do zdefiniowania odpowiednich reguł. W tym jednak miejscu pojawia się problem, ponieważ nie we wszystkich zagadnieniach można łatwo zdefiniować odpowiedni zestaw reguł. Powoduje to, że często korzysta się z narzędzi do automatycznego wyodrębniania reguł, takich jak drzewa decyzji (CART, C4.5) czy też algorytmy sekwencyjnego pokrywania (AQ, CN2 itp) [2]. Tego typu algorytmy same poszukują – uczą się odpowiednich zestawów reguł na podstawie zebranych danych doświadczalnych. Czasem jednak w systemach doradczych reprezentacja wiedzy w postaci klasycznych reguł jest niewystarczająca i konieczne jest skorzystanie z innych narzędzi. Ich przykładem są np. systemy reguł rozmytych. Celem ich powstania była integracja i ułatwienie implementacji wiedzy ekspertów dziedzinowych w postaci możliwej do przetwarzania przez systemy informatyczne. Cel ten osiągnięto poprzez sprzęgnięcie możliwości przetwarzania języka naturalnego, w tym nieprecyzyjnego przetwarzania informacji stosowanego przez ludzi, z odpowiednim zapisem matematycznym do postaci regułowej.

Poniżej zaprezentowano zestaw zadań laboratoryjnych poświęconych każdemu z powyższych zagadnień.

Klasyczne systemy ekspertowe

Klasyczne systemy ekspertowe to systemy, w których programowanie zwykle odbywa się na zasadzie opisania zachowania systemu poprzez zdefiniowanie odpowiednich reguł – stanowiących heurystyki opisujące zestaw akcji, jakie mają być podjęte w określonej sytuacji. Każda reguła składa się z części składowej „if” oraz z konkluzji, czyli słowa kluczowego „then”. Część warunkowa reguły (poprzednik reguły) określona jest za pomocą wzorców określających fakty, których wystąpienie powoduje uruchomienie reguły. System ekspertowy udostępnia specjalny mechanizm, zwany mechanizmem wnioskowania, którego zadaniem jest automatyczne dopasowanie faktów do wzorców i określenie reguł, które mogą zostać uruchomione.

Akcje związane z regułami, których przesłanki są spełnione są wywoływane w momencie, gdy system wnioskowania zostaje uruchomiony. Wówczas system wnioskowania wybiera regułę, a następnie wykonuje akcje związane z wybranymi regułami. Wykonanie akcji może modyfikować bazę danych poprzez dodanie lub usunięcie nowych faktów, a tym samym może doprowadzić do spełnienia przesłanek innych reguł. Wówczas system wnioskowania wybiera inną regułę i wywołuje jej akcje. Ta procedura wykonywana jest dopóki istnieją reguł gotowych do uruchomienia. [1]

Jednym z przykładów szkieletowych systemów ekspertowych jest narzędzie o nazwie CLIPS [7]. Projekt ten udostępnia kompletne środowisko do budowy systemu regułowego z uwzględnieniem koncepcji programowania obiektowego oraz proceduralnego. Środowisko CLIPS napisano w języku C (ang. C Language Integrated Production System), jednak posiada on bogate wsparcie również dla innych platform programistycznych jak np. dla środowiska Java.

[1] Na podstawie What is CLIPS „http://clipsrules.sourceforge.net/WhatIsCLIPS.html”

Środowisko CLIPS oferuje użytkownikowi wiersz poleceń umożliwiający w sposób interaktywny wprowadzanie poleceń oraz podgląd wyników działania systemu. Każde polecenie w systemie CLIPS wprowadzane jest w postaci słowa kluczowego umieszczonego w nawiasach okrągłych, wraz z odpowiednimi przełącznikami.

Zestaw operacji można podzielić na polecenia umożliwiające zarządzanie bazą faktów, zarządzanie bazą reguł oraz polecenia ogólnego przeznaczenia. Lista poleceń przedstawia poniższa tabela

Polecenia dotyczące faktów

(facts x)	wyświetla listę faktów począwszy od x
(assert (jablko)) (assert (fakt1) (fakt2) (fakt3))	dodaje nowy fakt/fakty
(reset)	resetuje listę faktów
(retract x)	usunięcie X'tego faktu, gdzie x to nr faktu
(deffacts nazwa „opis” (fakt1) (fakt2) ..)	definiuje listę faktów o określonej nazwie zawsze po wykonaniu polecenia (reset)
(undeffacts nazwa)	cofie definicję określonych faktów
(modify adres_faktu (nowa_wartość))	Umożliwia modyfikację wartości faktu
(deftemplate nazwa (slot nazwa1) (slot nazwa2) (multislot nazwa3))	pozwala na definiowanie schematu faktów (rekordu) który składa się z pól nazwa1, nazwa2, nazwa3, przy czym nazwa1 i nazwa2 mogą przyjmować tylko pojedynczą wartość, natomiast nazwa3 może przyjmować kilka wartości. Schemat faktu to struktura danych opisująca z jakich elementów powinien składać się określony fakt np. (deftemplate osoba (slot imie) (slot nazwisko) (slot kolor_oczu))
(ppfact adres_faktu)	wyświetla fakt o określonym adresie
(fact-index adres_faktu)	zwraca numer faktu
(save-facts nazwa_pliku)	zapisanie faktów do pliku
(load-facts nazwa-pliku)	wczytanie faktów z pliku

Polecenia dotyczące reguł

(rules)	wyświetla listę reguł
(defrule nazwa „komentarz” (przeslanka1) (przeslanka2) .. ⇒ (wniosek1) (wniosek2))	definiowanie reguły przy czym przeslankaX i wniosekX to najczęściej fakty, w szczególności wniosekX to (assert) (retract)
(undefrule nazwa)	usunięcie reguły
(ppdefrule nazwa)	umożliwia podgląd reguły
(refresh)	ponowne sprawdzenie aktywacji reguły, która została aktywowana. Normalnie reguły są aktywowane po zmianie stanu tabeli faktów, więc po jednokrotnej aktywacji reguły nie zostanie ona aktywowana więcej, chyba że wydamy komendę (refresh)
(matchs nazwa)	pozwala podejrzeć stan reguły, tzn które wzorce są aktywowane
(not (fakt))	przy definiowaniu reguł możemy używać negacji typu (defrule xxx (not (zaplon tak)) ⇒ (assert (zepsute auto)))
(or (fakt1) (fakt2) …)	Operacja logiczna „lub” związany z występowaniem fakt1 oraz fakt2 (jeżeli jeden z faktów wystąpi to wówczas wynik operacji jest prawdziwy)
(and (fakt1) (fakt2) …)	Operacja logiczna „i” związany z występowaniem fakt1 oraz fakt2 (oby wynik był pozytywny obydwa fakty muszą występować w bazie danych)

(test (typ fakt wartość))	pozwala na realizację testów typu > < itp
(save nazwa_pliku)	zapisuje reguły
(load nazwa_pliku)	wczytuje reguły

Polecenia ogólnego przeznaczenia

(clear)	czyści całą listę faktów i reguł
(watch facts) / (watch rules) /)(watch activations) (watch all)	włącza podgląd odpowiedniej bazy (watch facts) – podgląd bazy danych, (watch rules) – podgląd reguł, (watch activations) – podgląd aktywacji reguł (watch all) – podgląd wszystkich powyższych. Włączenie podglądu oznacza, że każda modyfikacja którejś z baz będzie uwidoczniona odpowiednim komunikatem
(unwatch)	przerwij podgląd np. (unwatch all) – przerwij podgląd wszystkiego, szczegóły patrz wyżej
(agenda)	wyświetla informacje o regule/regułach aktywowanej przez dane przesłanki
(run) / (run n)	uruchamia system wnioskowania / uruchamia pierwszych n-kroków systemu wnioskowania
?nazwa	tworzy zmienną o nazwie *nazwa*, można też użyć samego ? wówczas jest to dowolna zmienna (nie mamy do niej dostępu bo nie znamy nazwy) która jest wymagana
?zmienna ← (fakt)	przepisanie adresu faktu do zmiennej zmienna (zmienna ta zawiera adres w postaci np. fact-0), zmienna jest zmienną lokalną więc dostęp do niej jest np. wewnątrz reguł
$?	Symbol zastępczy, np. gdy któryś z slotów wzorca faktu ma być niezdefiniowany (może przyjmować dowolną wartość)
(bind zmienna wartość)	przypisanie wartości do zmiennej
~	Negacja
(read)	wczytanie danych przez użytkownika
(printout t „napis” crlf)	wypisanie tekstu na ekranie (terminalu „t”/konsoli) i przejście do nowej linii

System ekspertowy CLIPS stanowi implementację bazy reguł elementarnych, co oznacza, że w bazie faktów zapisywane są jedynie występujące fakty (będące prawdą). CLIPS nie posiada możliwości przechowywania zaprzeczonych faktów.

Przy wykorzystaniu systemów regułowych do podejmowania decyzji (zagadnienia klasyfikacji) zwykle dąży się do tego by reguły nie pokrywały się, co zapewnia niesprzeczność bazy reguł. Innymi słowy dążymy do tego, aby system nie podejmował decyzji wzajemnie się wykluczających. Cel ten można osiągnąć przez odpowiedni zapis reguł:

- listy reguł - lista reguł przetwarzana jest zgodnie z zasadą w której po kolei sprawdzana jest każda reguła w celu weryfikacji, czy jej wzorzec odpowiada faktom znajdującym się w bazie faktów. W przypadku języka CLIPS odpowiedni rezultat można osiągnąć bezpośrednio wpisując listę reguł. Należy jednak uważać na wykorzystanie operacji ***retract*** gdyż jej wydanie zmienia listę aktualnych faktów, co powoduje, że dla kolejnych reguł w bazie wiedzy dany fakt usunięty poleceniem ***retract*** nie będzie występował. Taki zapis bazy reguł jest nieodporny na występowanie pokrywania się reguł.
- Uporządkowana lista reguł – tego typu rozwiązania działa podobnie jak lista reguł, z tą różnicą, iż spełnienie jednej z reguł powinno zablokować wykonanie kolejnych reguł. Tego typu rozwiązanie stosuje się zwykle w sytuacji, gdy reguły wzajemnie się pokrywają. W warunkach środowiska CLIPS można to osiągnąć poprzez wydanie polecenia ***retract*** celem usunięcia faktów aktywujących określoną regułę.

- Drzewo decyzji – tego typu rozwiązanie pozwala na hierarchiczny zapis struktury reguł, tzn że spełnienie określonego warunku/warunków powinno zawężać obszar poszukiwań rozwiązania. Strukturę drzewa decyzji można zapisać w postaci regałowej na dwa sposoby:
    1. Reguła zapisywana jest bezpośrednio, jako ciąg przesłanek wymaganych do spełnienia warunku równoważnego określonemu liściowi w drzewie. Oznacza to, że liczba reguł jest równa liczbie liści w drzewie, a liczba przesłanek odpowiada głębokości drzewa dla danego liścia. W rezultacie otrzymany zestaw reguł redukuje się do dowolnej z wyżej opisanych struktur, gdyż warunki poszczególnych reguł wzajemnie wykluczają się.
    2. Drugie rozwiązanie bazuje na zapisywaniu reguł, jako ciągu dwóch przesłanek spełnienia przesłanki z poziomu wyżej (poziomu rodzica drzewa) oraz spełnienie określonego warunku związanego z danym węzłem. Takie rozwiązanie wymusza zwiększenie liczby zapisanych reguł, jednak ułatwia późniejszą modyfikację systemu, gdyż w przypadku konieczności dokonania zmian modyfikuje się jedynie wybrane pojedyncze reguły a nie cały zestaw reguł. Drugim zyskiem z takiego zapisu systemu reguł jest pozostawienie śladu w bazie faktów sposobu podejmowania decyzji.

  Przykłady obydwu systemów zapisu reguł drzew decyzji zamieszczono na wykładzie.

Zadania Lab1

Korzystając z powyżej określonych poleceń spróbuj rozwiązać poniższe zadania. W sprawozdaniu umieść odpowiednie polecenie niezbędne do rozwiązania poszczególnych zadań.

Zad. 1.

Utwórz fakt (dzisiaj jest sroda) – *(assert)*, następnie sprawdź czy fakt został utworzony *(facts)*. Ostatecznie usuń utworzony fakt – *(retract)*. Spróbuj dodać inne fakty typu wczoraj był… jutro jest ….

Zad. 2.

Zresetuj listę faktów *(reset).* Utwórz fakty będące inicjowane zawsze po restarcie systemu typu *(deffacts).* Sprawdź czy system działa poprawnie dodając jakieś fakty i resetując CLIPS'a

Zad. 3.

Utwórz regułę korzystając z polecenia *(defrule)*, która sprawdza czy dzisiaj jest środa, jeśli reguła jest spełniona powinien pojawić się nowy fakt *(jutro jest czwartek)*

Zad. 4.

Stwórz nową regułę w której przesłanką jest fakt (dzis jest sroda) wypisującą na ekranie komunikat *„Dziś jest środa"*

Zad. 5.

Reguła stworzona w Zad 3 i 4 działa tylko na dzień środa. Utwórz nową regułę „uniwersalną" wypisująca na ekranie komunikat dla dowolnego dnia tygodnia. W tym celu stwórz regułę wykorzystującą zmienne. Wykorzystując polecenie „defrule" utwórz regułę zastępując dzień tygodnia symbolem *?dzien*. Następnie w konkluzji wypisz na ekranie komunikat „Zmienna ma wartość" ?dzien.

Zad. 6.

W celu usunięcia określonego faktu możesz się również posłużyć zmiennymi. W tym celu korzystając z operatora <- możesz przepisać adres faktu do zmiennej, tak aby później móc wykorzystać polecenie (retract zmienna), które usunie niechciany fakt. Ponieważ tworzone zmienne mają zasięg lokalny, więc polecenie to jest do wykorzystania wewnątrz reguł, np.

chcąc usunąć z bazy faktów fakt odpowiedzialny za uruchomienie reguły możesz przy definicji reguły stworzyć zapis *„(…) ?zmienna<-(fakt) => (retract ?zmienna) (…)*” Opisz działanie takiej konstrukcji.

Zad. 7.

Spróbuj wykorzystać polecenie (read), tak aby dwa komunikaty w konkluzji reguły następowały dopiero po naciśnięciu enter

Zad. 8.

Stwórz system regułowy sterujący światłami drogowymi. System powinien odpowiednio przechodzić od stanu „światło zielone” do stanu „światło żółte”, „światło czerwone”, „światło żółto czerwone”, „światło zielone” , po czym powinien rozpoczynać cykl od początku.

Zad. 9.

Dokonaj ponownej implementacji systemu świateł jednak skorzystaj z polecenia deftemplate(). Wykorzystaj je do zdefiniowania struktury (wzorca) świateł np. w postaci

(deftemplate swiatla (slot czerwone)(slot zolte)(slot zielone))

Gdzie *deftemplate* deftemplate tworzy nam wzorzec węzła grafu reprezentującego sekwencję zmiany świateł.

Taki wzorzec możesz wykorzystać do tworzenia faktów. Np. przyjmując jako oznaczenie 0 dane statlo nie swieci się, oraz 1 – swiatło jest zapalone fakt utworzony za pomocą polecenia (assert (swiatla (czerwone 0)(zolte 0)(zielone 1))) oznacza że pali się swiatło zielone.

Podobnie tworząc regułę możesz użyć wzorców do weryfikacji stanu np. zmiana świateł z zielonych na żółte: (defrule zielone-zote ?a <-(swiatla (czerwone 0)(zolte 0)(zielone 1)) => (assert (swiatla (czerwone 0)(zolte 1) (zielone 0))) (retract ?a))

Taki zapis reguł w którym wykorzystuje się wzorce jest znacznie łatwiejszy do interpretacji.

Zadania Lab2

Zad. 1.

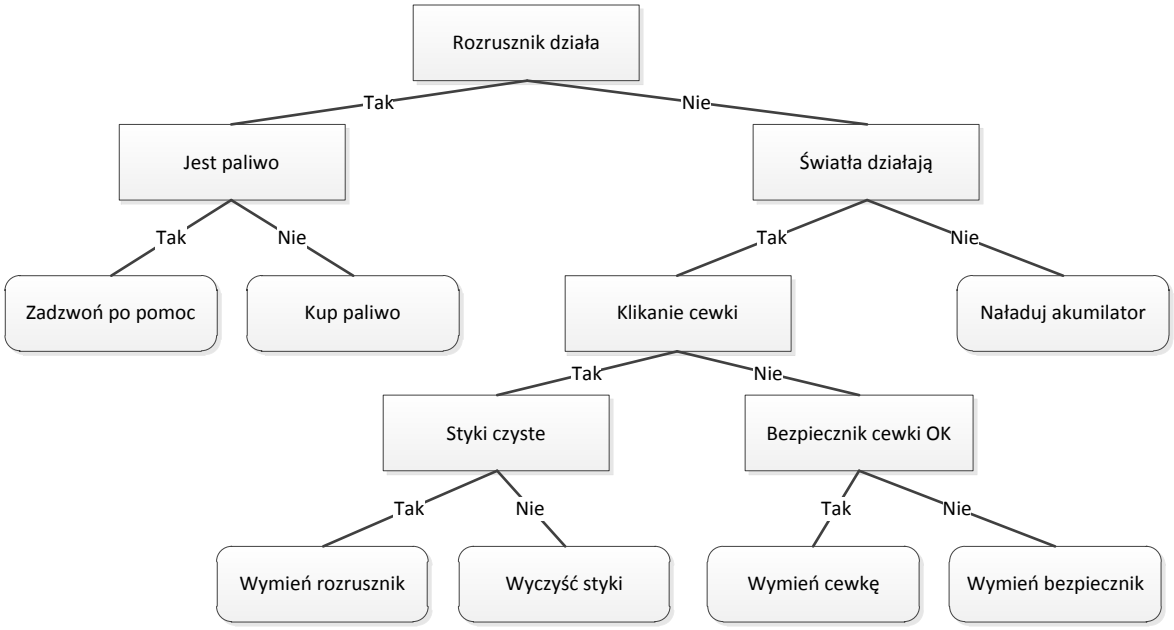
Zaimplementuj drzewo decyzji pokazane na Rys. 2 reprezentujące fragment sekcji diagnozowania awarii samochodu. W poniższym drzewie każdy zaokrąglony prostokąt odpowiada diagnozie problemu, natomiast każdy prostokąt odpowiada weryfikacji określonego warunku, a tym samym zadania pytania do diagnosty o stan określonej funkcjonalności.

Do realizacji zadania posłuż się poleceniem *(deffunction …)* służącym do tworzenia nowych funkcji. Odpowiednia funkcja powinna mieć postać:

```
(deffunction ask-yes-or-no (?question)
  (printout t ?question " (Tak/Nie) ")
  (bind ?answer (read))
  (while (and (neq ?answer tak) (neq ?answer nie))
    (printout t ?question " (Tak / Nie) ")
    (bind ?answer (read)))
  (return ?answer))
```

Funkcja prosi o udzielenie odpowiedzi *Tak* lub *Nie.* W tym celu wyświetla na ekranie odpowiedni komunikat i prosi o udzielenie odpowiedzi. Jeśli odpowiedź jest niepoprawna, to prosi o udzielenie odpowiedzi jeszcze raz.

W zadaniu do reprezentacji grafu wykorzystaj funkcję *(deftemplate …)* która powinna reprezentować ogólny węzeł w poniższym grafie, wraz z wszystkimi jego stanami.



Rys. 2 Schemat drzewa decyzji diagnozowania samochodu

Zad. 2.

W identyczny sposób jak wyżej najpierw zaproponuj graf pozwalające na diagnozowanie problemów z siecią komputerową, a następnie zaimplementuj zaproponowane drzewo. Budując drzewo uwzględnij możliwe problemy z siecią i ich diagnozowanie w oparciu o wiersz poleceń (zaproponuj odpowiednie polecenia wiersza poleceń umożliwiające weryfikację poprawności działania sieci). Lista niezbędnych czynników do weryfikacji (dobierz sam odpowiednią kolejność weryfikacji poszczególnych elementów działania sieci):

- brak karty sieciowej,
- weryfikacja poprawności bramy
- weryfikacja poprawności serwerów DNS
- kabel sieciowy odłączony,
- diagnozowanie miejsca występowania utraty pakietów
- poprawność konfiguracji adresu IP ,
- poprawność konfiguracji stosu TCP/IP,
- w przypadku serwera korzystania z serwera DHCP odnów dzierżawę adresu