Politechnika Warszawska

MSI

4. Wnioskowanie w logice

Włodzimierz Kasprzak

Układ

1. Twierdzenia o dedukcji
2. Uogólniona reguła *odrywania*
3. Wnioskowanie *„w przód” i „wstecz”*
4. Rezolucja w logice predykatów
5. Realizacja celu
6. Systemy ekspertowe
7. PROLOG
8. System regułowy
9. Sieci semantyczne (ramy)

1. Twierdzenia o dedukcji

- **W logice pierwszego rzędu zachodzą oba** znane nam już **twierdzenia o dedukcji**:

  $KB \models \alpha$ wtw. gdy $(KB \Rightarrow \alpha)$ **jest tautologią.**

  $KB \models \alpha$ wtw. gdy **formuła** $(KB \wedge \neg\alpha)$ **jest niespełnialna**.

- Potencjalnie nieprzeliczalna liczba wartościowań dla badanej teorii ogranicza rozstrzygalność procesu wnioskowania w tej logice.

Własność logiki predykatów - zachodzi twierdzenie:

- Problem stwierdzenia tego, czy konkretna formuła wyprowadzenia $(KB \Rightarrow \alpha)$ przedstawia wynikanie formuły zapytania ze zbioru prawdziwych formuł (tzn. z bazy wiedzy) czy nie, jest problemem **nierozstrzygalnym**.
- Jest on natomiast **pół-rozstrzygalny**: „istnieje algorytm, który dla zadanej formuły wyprowadzenia *A* stwierdza, że *A* jest tautologią, pod warunkiem, że tak istotnie jest.”

2. Uogólniona reguła odrywania

Uogólniona reguła odrywania (ang. *General Modus Ponens* – GMP):

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\mathrm{SUBST}(\theta, q)}$$

gdzie $(p_1', \ldots, p_n', p_1, \ldots, p_n, q)$ są **literałami**, a θ jest **podstawieniem** takim, że dla każdego $i = 1,2,\ldots,n$:

$$\mathrm{SUBST}(\theta, p') = \mathrm{SUBST}(\theta, p).$$

Np. $p_1' = Król(Jan)$, $p_1 = Król(x)$, $p_2' = Chciwy(y)$, $p_2 = Chciwy(x)$,

$q = Zło(x)$ → $\theta = \{x/Jan, y/Jan\}$, $\mathrm{SUBST}(\theta, q) = Zło(Jan)$

Zastosowanie GMP poprzedzone jest zamianą formuł w bazie wiedzy do postaci **kanonicznej Horna**:

- jest nią literał lub implikacja, w której lewą stronę stanowi koniunkcja pozytywnych literałów a prawą – pojedynczy pozytywny literał.

Uzgadnianie zmiennych (unifikacja)

- Podstawienie θ jest **unifikatorem** wyrażeń $E_1,...,E_n$ jeśli $\text{SUBST}(\theta, E_1) = ... = \text{SUBST}(\theta, E_n)$.

  Np. podstawienie $\{x/A\}$ jest unifikatorem wyrażeń $P(x)$, $P(A)$. Ale wyrażenia $P(x)$, $Q(b)$ nie są unifikowalne.

- Podstawą reguły GMP jest algorytm unifikacji $\text{UNIFY}(p,q)$, który zwraca **podstawienie** (tzw. najogólniejszy unifikator) dla literałów p, q, które czyni je identycznymi, lub zwraca *fail*, jeśli podstawienie nie istnieje.

- **Złożeniem podstawień** θ_1, θ_2 jest podstawienie $\text{COMPOSE}(\theta_1, \theta_2)$, które polega na wykonanie po kolei obu podstawień, czyli:

  $\text{SUBST}(\text{COMPOSE}(\theta_1, \theta_2), E) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, E))$.

Problem uzgadniania zmiennych

- Podstawienie θ jest **naogólniejszym unifikatorem** dla $\{E_1,..., E_n\}$, jeśli dla każdego unifikatora tych wyrażeń γ istnieje podstawienie λ takie, że $\gamma = \text{COMPOSE}(\theta, \lambda)$.
- Unifikowalne wyrażenia posiadają dokładnie jeden najogólniejszy unifikator.
- Przykład. Wyrażenia $P(\text{Jan}, \text{x})$, $P(\text{y}, \text{z})$ posiadają nieskończenie wiele unifikatorów:

  $\{\text{y/Jan, x/z}\}$, $\{\text{y/Jan, x/z, w/Fred}\}$,

  $\{\text{y/Jan, x/Jan, z/Jan}\}$

  Najogólniejszy unifikator: $\{\text{y/Jan, x/z}\}$
- **Zadanie uzgadniania zmiennych:** należy stwierdzić, czy dane wyrażenia $E_1,..., E_n$ są unifikowalne. Jeśli tak to należy znaleźć ich najogólniejszy unifikator.

Uzgadnianie zmiennych - przykład

- Problem unifikacji jest rozstrzygalny.
- Zakładamy, że dysponujemy funkcją *UNIFY*, która dla zadanych wyrażeń E_1, ... , E_n, zwraca ich najogólniejszy unifikator θ lub specjalną stałą *fail*, jeśli unifikator nie istnieje.

Przykład działania funkcji *UNIFY*:

p	q	θ
Zna(Jan,x)	Zna(Jan,Jane)	{x/Jane}
Zna(Jan,x)	Zna(y,OJ)	{x/OJ, y/Jan}
Zna(Jan,x)	Zna(y,Matka(y))	{y/Jan, x/Matka(Jan)}
Zna(Jan,x)	Zna(x,OJ)	{*fail*}

Przemianowanie zmiennych

- **Wariant formuły** (*przemianowanie zmiennych*): formuła A jest wariantem formuły B, jeśli A można otrzymać z B zastępując niektóre zmienne w B nowymi zmiennymi nie występującymi w B.
- Np. *Lubi* (x, Jan) jest wariantem formuły *Lubi* (y,Jan)
- Formuły naszej bazy wiedzy są takiej postaci, że każda z nich może zostać zastąpiona swoim wariantem (przemianowana) i otrzymamy bazę wiedzy równoważną z poprzednią postacią.

Standaryzacja rozłączna

- Dlaczego należy przemianować zmienne w formule przed wstawieniem jej do bazy wiedzy?

Załóżmy, że w KB istnieją formuły: $Zna(x, Elżbieta)$,

$$Zna(Jan, x) \Rightarrow Nienawidzi(Jan, x) .$$

Powinniśmy wywnioskować, stosując ogólną regułę odrywania, że $Nienawidzi(Jan, Elżbieta)$. Nie jest to możliwe – formuły $Zna(Jan, x)$ i $Zna(x, Elżbieta)$ nie są unifikowalne.

Dopiero, gdy przemianujemy jedną ze zmiennych (np. formułę $Zna(x, Elżbieta)$ zastąpimy przez $Zna(y, Elżbieta)$) to będziemy mogli wyprowadzić: $Nienawidzi(Jan, Elżbieta)$.

- **Możemy założyć**: przesłanki ogólnej reguły odrywania **nie zawierają** wspólnych zmiennych, gdyż formuła została wcześniej poddana standaryzacji rozłącznej zmiennych.

Wstawianie formuły do bazy wiedzy

Formuła wprowadzana do KB jest najpierw przekształcana:

1. Standaryzacja rozłączna zmiennych – każdy kwantyfikator związany jest unikalną zmienną.
2. Eliminowane są kwantyfikatory egzystencjalne (skolemizacja)
3. Opuszczane są uniwersalne kwantyfikatory.
4. Formuła jest przekształcana do postaci normalnej (Horna lub CNF).
5. Dodawanie do bazy wiedzy:

- Standaryzacja rozłączna zmiennych formuły względem bazy wiedzy
- Formuła o postaci koniunkcji klauzul Horna lub postaci CNF jest przekształcana do zbioru klauzul (reguła eliminacji koniunkcji).

3. Wnioskowanie „w przód” i wstecz

W rachunku zdań procedura wnioskowania sprawdza jedynie, czy zadana formuła wynika z bazy wiedzy.

Np. **Ask**(KB, *Brat*(Jan,Piotr)) , **Ask**(KB, *Lubi*(Anna,Jan)).

W logice predykatów możemy spytać bazę wiedzy o prawdziwość formuły dla wielu obiektów na raz.

Mając formułę S pytamy się, czy istnieje podstawienie θ dla którego S wynika z bazy wiedzy.

- **Ask**(KB, S) zwraca wszystkie podstawienia θ takie, że

$$KB \models SUBST(\theta, S) .$$

- Pytania do KB są typu: kto, gdzie, kiedy?

  Np.: „*Kto jest bratem Piotra*?”

  Ask(KB, ∃x Brat(x, Piotr))

  Oczekiwana odpowiedź, np.: {{x/Jan}, {x/Stefan}} .

Przykład zapytań dla „świata Wumpusa”

W „świecie Wumpusa” korzystamy z bazy wiedzy w logice predykatów:

- Typowa obserwacja w chwili t:

Percepcja([smród, wiatr, błysk, uderzenie, krzyk], t)

Np. agent czuje smród i wiatr (i brak innych) w chwili $t = 5$:

Tell(KB, **TwórzZdanieObserwacji**(

Percepcja([smród, wiatr, *brak*, *brak*, *brak*], 5)))

- Jaka najlepsza akcja w chwili $t = 5$ wynika z KB?

Funkcja **TwórzZapytanieAkcji(5)** konstruuje pytanie typu

$\exists$a *NajlepszaAkcja*(a, 5)):

Ask(KB, $\exists a$ *NajlepszaAkcja*(a, 5))

Odpowiedź: $\theta = \{a / \text{Strzał}\}$, co oznacza, że z KB wynika:

SUBST(θ, S) = *NajlepszaAkcja*(Strzał, 5).

Wnioskowanie „w przód” (FC_ASK)

Stosujemy **wnioskowanie w przód** zasadniczo wtedy, gdy nowa formuła p zostaje dodana do bazy wiedzy KB (implementujemy to jako podfunkcję funkcji **TELL**). Przykładową implementacją jest procedura FC_TELL() podana na kolejnej stronie.

Przy takim rozwiązaniu, gdy funkcja agenta wysyła zapytanie (w ramach funkcji **ASK**) o wynikanie z KB pewnej formuły $q(\boldsymbol{Z})$, (gdzie Z jest listą zmiennych w formule) implementacja **wnioskowania w przód** polega jedynie na sprawdzeniu możliwości uzgodnienia (unifikacji) formuły $q(\boldsymbol{Z})$ z formułami istniejącymi w KB. Np.:

function FC_ASK(KB, $q(\boldsymbol{Z})$) **return** [$wynik$, θ]
begin $wynik$ = **False**, $\theta = \varnothing$
  for (każda formuła q' w KB unifikowalna z q) **do**
    $wynik$ = True; ρ = UNIFY(q', q); $\theta = \theta \cup \rho_{|Z}$;
  end
  return [$wynik$, θ];
end

Wnioskowanie „w przód” (FC_TELL)

Procedura **wnioskowania w przód,** stanowiąca element funkcji TELL, korzysta z reguły „uogólnione modus ponens” i wyprowadza ona wszystkie nowe zdania, które wynikają z $KB \cup \{ p \}$:

procedure FC_TELL(KB, p)
begin
  if (w KB istnieje już wariant formuły p) **then return**;
  Dodaj p do KB;
  for $((p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q) \in KB$ takie, że zachodzi unifikacja
    $\exists i$ UNIFY(p_i, p) = θ) **do**
    FC_WNIOSKUJ(KB, [p_1,..., p_{i-1},p_{i+1},…,p_n], q, θ);
  end
end

Rekursywna procedura FC_WNIOSKUJ() uzgadnia, jeśli to możliwe, dalsze klauzule w poprzedniku formuły. Jeśli wszystkie one zostaną uzgodnione, to wyprowadzone zostają nowe zdania (jedno lub więcej, w zależności od liczby możliwych podstawień) - dla każdego nowego zdania wywoływana jest procedura początkowa.

Wnioskowanie „w przód” (TELL) (2)

```
procedure FC_WNIOSKUJ(KB, warunki, konkluzja, θ)
begin
        if (warunki == []) then
            FC_TELL(KB, SUBST(θ, konkluzja));
        else
            for (każde p' ∈ KB takie, że
                    θ2 = UNIFY(p', SUBST(θ , Pierwszy(warunki))))
                    do
                    θ  = COMPOSE(θ, θ2);
                    FC_WNIOSKUJ(KB, Reszta(warunki), konkluzja,θ);
            end
        end
end
```

Wnioskowanie wstecz (1)

Wnioskowanie wstecz stosujemy wtedy, gdy chcemy otrzymać odpowiedź na pytanie zadane do bazy wiedzy (implementujemy je w ramach funkcji **ASK**).

function BC_ASK(KB, $q(\boldsymbol{Z})$) **return** [$wynik$, $\theta_{\backslash \boldsymbol{Z}}$]
// $wynik$ = **True/False**, $\theta_{\backslash \boldsymbol{Z}}$ = zbór podstawień pod zmienne z listy $\boldsymbol{Z}$;
begin $\theta = \{\}$;
$wynik$ = BC_LIST(KB, [$q(\boldsymbol{Z})$], θ);
return [$wynik$, $\theta_{\backslash \boldsymbol{Z}}$];
end

Parametry funkcji BC_LIST() to: baza wiedzy KB, lista celów $[q]$, szukany zbiór podstawień θ. Jest to funkcja rekursywna, wywoływana najpierw dla celu (formuły zapytania) a następnie dla kolejnych podcelów, przy jednoczesnym rozszerzaniu zbioru podstawień.

Wnioskowanie wstecz (2)

function BC_LIST(*KB*, *cele*, θ) : **return** *wynik*
// a także odpowiednio rozszerza zbiór podstawień θ);
begin
wynik $\leftarrow \varnothing$;
if (*cele* są puste) **then return** $\{\theta\}$;
q$\leftarrow$SUBST(θ, Perwszy(cele));
for (każdy $r \in$ KB taki, że STAND-ROZŁĄCZNA(r) = ($p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q'$) $\in$ *KB* i zachodzi $\theta' \leftarrow$UNIFY(q, q')) **do**
wynik $\leftarrow$ BC_LIST(*KB*, SUBST(θ‘, [p_1,...,p_n| Reszta(*cele*)], COMPOSE(θ, θ')) $\cup$ *wynik*;
return *wynik*;
end

Własności wnioskowania wstecz

- Potencjalnie niepełne z powodu istnienia nieskończonych pętli:

  $\Rightarrow$ zabezpieczenie: sprawdzanie aktualnego celu z każdym celem pamiętanym na stosie
- Potencjalnie nieefektywne na skutek powtarzania pod-celów (zarówno pozytywnych jak i negatywnych)

  $\Rightarrow$ zabezpieczenie: pamiętanie poprzednich wyników (dodatkowa pamięć)
- Stosowane w programowaniu w logice (dla postaci kanonicznej Horna).

4. Rezolucja w logice predykatów

- Pełna wersja reguły rezolucji w logice predykatów :

$$\frac{l_1 \vee \cdots \vee l_k, \qquad m_1 \vee \cdots \vee m_n}{\mathrm{SUBST}(\theta, (l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n))}$$

  gdzie $\mathrm{UNIFY}(l_i, \neg m_j) = \theta$.

- Zakłada się, że formuły są w postaci klauzul Horna, tzn. l_1 , m_1 są literałami a dwie formuły warunkujące regułę zostały standaryzowane rozłącznie, tzn. nie mają wspólnych zmiennych.

- Np.

$$\frac{\neg Bogaty(x) \vee Nieszczęśliwy(x) \,, \; Bogaty(Jan)}{Nieszczęśliwy(Jan)}$$

  gdzie $\theta = \{x/Jan\}$

- **Algorytm wnioskowania przez rezolucji** stosuje się do formuł w postaci CNF („conjunctive normal form”).

Konwersja formuły do CNF

Np. „*każdy kto kocha wszystkie zwierzęta jest kochany przez kogoś*”:

$$\forall x\,[\forall y\; Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y\; Loves(y, x)]$$

Kroki konwersji

1. Standaryzacja rozłączna zmiennych: każdy kwantyfikator korzysta z innej zmiennej.

$$\forall x\,[\forall y\; Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists z\; Loves(z, x)]$$

2. Przesuwamy kwantyfikatory w lewo

$$\forall x\; \forall y\; \exists z\,[Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [Loves(z, x)]$$

3. Skolemizacja: każda egzystencjalna zmienna (i jej kwantyfikator) jest zastępowana przez **funkcję Skolema** dla zmiennej należącej do obejmującego kwantyfikatora uniwersalnego:

$$\forall x\; \forall y\,[Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [Loves(F(x, y), x)]$$

Konwersja do CNF (2)

4. Pomijamy uniwersalne kwantyfikatory:

$$(Animal(y) \Rightarrow Loves(x, y)) \Rightarrow (Loves(F(x, y), x))$$

5. Eliminacja (ewentualnej równoważności i) implikacji:

$$\neg(\neg Animal(y) \vee Loves(x,y)) \vee Loves(F(x, y), x)$$

6. Przesuwamy $\neg$ w prawo:

$$(\neg\neg Animal(y) \wedge \neg Loves(x,y)) \vee Loves(F(x, y), x)$$

$$(Animal(y) \wedge \neg Loves(x,y)) \vee Loves(F(x, y), x)$$

7. Rozdzielamy $\vee$ względem $\wedge$:

$$(Animal(y) \vee Loves(F(x, y), x)) \wedge (\neg Loves(x, y) \vee Loves(F(x, y), x))$$

8. Ewentualnie z klauzul usuwamy każdy literał $\neg\mathrm{True}$ i False.

9. Ewentualnie usuwamy klauzule zawierające literał $\neg\mathrm{False}$ lub True.

Wnioskowanie przez zaprzeczenie

Procedury wnioskowania (dowodzenia) formuł stosujące regułę rezolucji prowadzą **dowód przez zaprzeczenie**, tzn. aby dowieść, że

$$KB \models \alpha$$

pokazują one, że zdanie

$$(KB \wedge \neg\alpha)$$

jest niespełnialne.

Jeśli w procesie wnioskowania przez rezolucje zostanie wygenerowane „zdanie puste” to oznacza to, iż zapytanie jest spełnione przy takim wartościowaniu zmiennych w zapytaniu, które prowadzi do zdania pustego.

Jeśli nie można wygenerować dalszych rezolwent, a nie powstało zdanie puste to zdanie zapytania nie jest spełnione w aktualnym modelu formuł w bazie wiedzy.

Rezolucja w logice predykatów

function RESOLUTION(*KB*, $q(\boldsymbol{Z})$) **return** [*wynik*, $\theta_{\backslash \boldsymbol{Z}}$]
// *wynik* = **True/False**, $\theta_{\backslash \boldsymbol{Z}}$ = zbór podstawień pod zmienne z listy $\boldsymbol{Z}$;
begin *wynik* = **False**; $\theta_{\backslash \boldsymbol{Z}} = \varnothing$; $\theta = \varnothing$;
formuły = *KB* $\cup$ { zanegowana formuła *q* }; nowe = *formuły*;
REPEAT
aktualne = $\varnothing$;
FOR każda formuła *p* w *nowe* **DO**
FOR każda formuła *p*' w { *formuły* - *p*} **DO**
[***rezolwenty***, θ] = KROK-REZOLUCJI(*p*, *p*');
IF (θ <> „*fail*") & ($\varnothing \in$ ***rezolwenty***) **THEN**
{ **wynik** = True; $\theta_{\backslash \boldsymbol{Z}}$= COMPOSE($\theta_{\backslash \boldsymbol{Z}}$,θ) ; }
aktualne = *aktualne* $\cup$ ***rezolwenty ;***
formuły= *formuły* $\cup$ aktualne; *nowe* = *aktualne*;
UNTIL *aktualne* < > $\varnothing$
return [wynik, $\theta_{\backslash \boldsymbol{Z}}$];
end

Rezolucja w logice predykatów (2)

Baza wiedzy powinna być niesprzeczna i zawierać formuły o postaci CNF. Lista szukanych podstawień $\theta_{|Z}$ jest początkowo pusta.

W pierwszej iteracji zbiór $nowe(1)$ tworzą wszystkie podformuły o postaci alternatyw literałów (klauzul) pochodzące z KB i negacji zapytania. Zbiór $aktualne(1)$ jest początkowo pusty.

W i-tej iteracji wykonywane są kroki rezolucji dla każdej klauzuli p ze zbioru $nowe(\mathrm{i})$ z każdą klazulą, pochodzącą ze zbioru $\{formuły(\mathrm{i}) - p\}$. Nowe rezolwenty (nie wygenerowane wcześniej) są dodawane do zbioru $aktualne(\mathrm{i})$.

Jeśli w danym kroku została wyprowadzona klauzula pusta (jedno- lub wielokrotnie) to nastąpiło to przy podstawieniu (-ach) θ pod zmienne tych formuł. Funkcja wnioskowania zapamiętuje wszystkie podstawienia pod zmienne zapytania w zbiorze $\theta_{|Z}$ które dały wynik pozytywny wnioskowania.

Rezolucja w logice predykatów (3)

Jednak proces wnioskowania jest kontynuowany dla znalezienia ewentualnie dalszych podstawień, przy których generowana jest kolejna formuła pusta.

Proces wnioskowania kończy się wtedy, gdy nie można już wygenerować żadnej nowej rezolwenty. Jeśli w żadnej z wykonanych iteracji głównej pętli funkcji nie wygenerowano formuły pustej to zwracany jest wynik „False”. W przeciwnym razie wynik jest „True” a zbiór podstawień $\theta_{|Z}$ *jest niepusty.*

Uwaga

Jeśli w zapytaniu NIE ma zmiennych to traktujemy je jak zapytanie dla rachunku zdań, czyli odpowiadamy jedynie False/True a zbiór $\theta_{|Z}$ pozostaje pusty.

W takiej sytuacji możemy też przerwać proces wnioskowania po wykryciu pierwszej klauzuli pustej.

5. Realizacja celu

Działanie agenta w „świecie Wumpusa” ma charakter **celowy :**

- Cel 1: Wybierając najlepszą (najbezpieczniejszą lub najtańszą) akcję w danej sytuacji, agent potrafi rozsądnie poruszać się po jaskini w **celu dotarcia do złota**. Nie ma tu jawnej **lokalizacji miejsca** lecz jedynie warunek wykonania najlepszej akcji – „pobierz złoto jeśli to możliwe”. Agent poznaje środowisko i bezpiecznie przemieszcza się dopóki nie znajdzie złota.
- Cel 2: Po „pobraniu złota” kolejnym **celem** agenta staje się bezpieczny powrót do kwadratu startowego i wydostanie się z jaskini. Cel zostaje jawnie określony w postaci własności dodanej do bazy wiedzy w odpowiedniej sytuacji:

$$\forall_{\mathbf{x},s} \; TrzymaZłoto(\mathbf{x}, s) \Rightarrow LokalizacjaCelu([1,1], s)$$

Techniki realizacji celu

Rozwiązanie problemu postrzegamy abstrakcyjnie jako sekwencję akcji realizującą zadany cel. Trzy główne techniki:

- **Wnioskowanie -** ogólna technika, ale najmniej efektywna z trzech - polega na ciągłym zadawaniu do bazy wiedzy zapytania o najlepszą akcję w danej sytuacji.
- **Przeszukiwanie** przestrzeni stanów problemu - ujawniana jest wiedza o powiązaniach pomiędzy stanami środowiska i wykonywanymi akcjami. Wybór akcji jest efektem realizacji odpowiedniej strategii przeszukiwania przestrzeni stanów.
- **Planowanie działań -** efektywny wybór akcji wtedy, gdy przestrzeń stanów problemu jest bardzo duża. Polega na wyznaczeniu sekwencji akcji w oparciu o dodatkową informację. Np. ujawniane są własności poszczególnych stanów i określany jest sposób oddziaływania każdej akcji na te własności, co pozwala skupić się na akcjach relewantnych do danej sytuacji.

Wnioskowanie jako realizacja celu

Podsystem sterowania posługuje się jedynie mechanizmem wnioskowania dla wyboru akcji:

```
funkcja KBsystemZWyboremAkcji(obserwacja)  zwraca: akcja
{ static: KB, // baza wiedzy
          t; // licznik czasu, początkowo wynosi 1
  TELL(KB, UtwórzZdanieObserwacji(obserwacja, t));
  for each (akcja in ListaMożliwychAkcji(KB, t)) {
        if (ASK(KB, UtwórzZapytanieAkcji(t, akcja) ) {
              t ← t+1;
              TELL(KB, UtwórzZdanieAkcji(akcja, t));
              return akcja;
        }
}
```

6. Systemy ekspertowe

System ekspertowy można zaimplementować w postaci systemu z bazą wiedzy – dysponuje językiem deklaratywnej reprezentacji wiedzy i mechanizmem wnioskowania.

- Logiczne systemy ekspertowe – przykładem jest PROLOG.
- Systemy regułowe (np. OPS5, CLIPS, SOAR) – stosują formuły implikacji dla reprezentacji warunków wykonania i opisu akcji; wśród akcji są operacje wprowadzania i usuwania do/z bazy wiedzy i operacje we/wy; stosują **wnioskowanie w przód**.
- „Ramy" (*frames*) i sieci semantyczne (np. SNePS, Netl, KL-ONE) – obiekty są węzłami grafu a ich relacje binarne reprezentują zależności typu: „*jest częścią*", „*jest specjalizacją*"; możliwe jest wnioskowanie **w przód** i **wstecz.**

Logiczny system wnioskowania

Przykłady logicznych systemów wnioskowania

- Dowodzenie twierdzeń (*theorem provers*) (np. AURA, OTTER) – stosują **rezolucję** w logice predykatów dla dowodzenia twierdzeń w zadaniach matematycznych oraz do realizacji systemu zapytań do bazy wiedzy.
- Języki programowania w logice (np. Prolog) – zwykle stosują **wnioskowanie wstecz**, nakładają ograniczenia na język logiki, dysponują proceduralnymi elementami we/wy.

Zadania systemu wnioskowania

Zadania realizowane przez system logicznego wnioskowania:

1. WPROWADŹ – operacja wprowadzania nowego faktu do bazy wiedzy – realizacja w postaci funkcji TELL;
2. WNIOSKUJ nowe fakty z połączenia dotychczasowej zawartości bazy wiedzy i nowo wprowadzonych faktów – realizacja w systemie opartym o wnioskowanie wprzód jako część funkcji TELL;
3. DECYDUJ czy zapytanie WYNIKA z bazy wiedzy – realizacja w postaci funkcji ASK;
4. DECYDUJ czy zapytanie ISTNIEJE w bazie wiedzy – ograniczona wersja funkcji ASK;
5. USUŃ – usuwa zdanie z bazy wiedzy z różnych przyczyn – zdanie jest nieprawdziwe, już niepotrzebne lub modyfikacja wynika ze zmiany w środowisku.

Elementy procedur dostępu

O efektywności procedur wnioskowania decyduje głównie właściwa implementacja bazy wiedzy i tych części funkcji TELL i ASK, które bezpośrednio odwołują się do KB.

- Implementacja zdań i formuł w KB – Np. wprowadzamy bazowy typ danych COMPOUND:

  - reprezentuje powiązanie operatora (czyli predykatu, funkcji lub spójnika logicznego) z listą argumentów (czyli z termami lub zdaniami);

  - posiada pola dla reprezentacji operatora (OP) i argumentów (ARGS).

  Np. istnieje c: $P(x) \wedge Q(x)$. Wtedy OP[c] = $\wedge$ i ARGS[c]=[P(x), Q(x)].

- STORE(KB,S) i FETCH(KB,Q) – funkcje dostępu do KB:

  - przy nieuporządkowanej bazie wiedzy złożoność obliczeniowa obu funkcji wynosi $O(n)$, gdzie n- liczba elementów KB;

  - indeksowanie z wykazem asocjacyjnym;

  - indeksowanie w postaci drzewa.

- Algorytm UNIFIKACJI.

7. Prolog

Język programowania logicznego umożliwia implementację logicznego systemu wnioskowania dzięki temu, że:

- posiada reprezentację logicznych formuł,
- umożliwia dołączanie informacji sterującej procesem wnioskowania,
- posiada mechanizm wnioskowania.

Typowym przedstawicielem takich języków jest **PROLOG**.

- Program w Prologu jest zbiorem klauzul Horna - czyli każda formuła jest postaci formuły atomowej lub implikacji, gdzie w poprzedniku występują dodatnie literały a następnik jest pojedynczym literałem.
- Dużymi literami oznacza się zmienne, małymi – stałe.
- Prawdziwość predykatów sprawdzana jest za pomocą dołączonego kodu programu a nie w wyniku wnioskowania.

Klauzule w Prologu

- W Prologu stosuje się zapis kanoniczny Horna w formacie implikacji pisanej w odwrotnej kolejności (następnik implikacji poprzedza poprzednika implikacji) a literały poprzednika są niejawnie połączone koniunkcją, czyli:

  nagłówek **:-** literał$_1$, … literał$_n$.

  Np.:

  przestępca(X) **:-** *Amerykanin*(X), *broń*(Y), *sprzedaje*(X,Y,Z), *wrogi*(Z).

- Każda klauzula Horna w Prologu jest jednej z postaci:

  $\neg P_1 \vee \ldots \vee \neg P_n \vee Q$ (*równoważne:* $(P_1 \wedge \ldots \wedge P_n) \Rightarrow Q$)

  Q (*formuła atomowa – literał*)

  $\neg P_1 \vee \ldots \vee \neg P_n$ (*klauzula celu*)

  $\perp$ (*klauzula pusta*)

Klauzule w Prologu

- W Prologu podane poprzednio klauzule Horna zapisujemy:

  Q :- $P_1, \ldots, P_n$.

  Q :- . (*formuła atomowa – literał*)

  :- $P_1, \ldots, P_n$. (*klauzula celu*)

  $\perp$. (*klauzula pusta*)

Klauzule Horna posiadają swoje **interpretacje w programie**:

- Klauzulę postaci Q :- $P_1, \ldots, P_n$. interpretujemy jako procedurę o nagłówku Q i treści $P_1, \ldots, P_n$. Wywołanie procedury Q sprowadza się do kolejnego wywołania procedur $P_1, \ldots, P_n$.
- Klauzulę Q :- . interpretujemy jako procedurę o pustej treści.
- Klauzulę :- $P_1, \ldots, P_n$. traktujemy jako dane programu.
- ?- $P_1, \ldots, P_n$. – zapytanie.
- Klauzulę pustą traktujemy jako instrukcję *stop*.

Predykaty w Prologu

- Wbudowane predykaty operacji arytmetycznych.

  Np. X **is** 4+3 – formuła jest prawdziwa, gdy zmienna X posiada wartość 7.

  5 **is** X+Y – prawdziwość takiej formuły nie może zostać sprawdzona na gruncie wbudowanych operacji.

- Wbudowane predykaty „operacji specjalnych” (np. WE/WY, przypisania).
- Założenie o zupełności świata - symbol "negacji” nie jest używany do tworzenia zanegowanych formuł, lecz odpowiada operacji **not** („nieprawda, że w aktualnej interpretacji zachodzi dany fakt”):

  Np. dla klauzuli: *alive*(X) :- **not** *dead*(X) zajdzie konkretny fakt *alive*(*Joe*) jeśli wykażemy, że *dead*(*Joe*) jest nieprawdziwe.

Wnioskowanie w Prologu

Program w Prologu to uporządkowany ciąg procedur.

- **Dane** to klauzula celu.
- **Wykonanie programu** to realizacja wnioskowania metodą rezolucji (lub metodą wstecz).

Programowanie w Prologu ma zasadniczo charakter **deklaratywny** (kod programu jest logiczną specyfikacją problemu a jego wykonanie – wnioskowaniem logicznym).

Jednak w celu zwiększenia efektywności występuje też wiele **mechanizmów pozalogicznych** :

- duży zbiór wbudowanych predykatów związanych z arytmetyką i wejściem/wyjściem – sprawdzenie poprawności tych predykatów polega na wykonaniu odpowiedniego kodu procedur a nie na wnioskowaniu logicznym;
- szereg funkcji systemowych i obsługujących bazę wiedzy.

Prolog – przykład zapytania

- Niech append oznacza operację połączenia dwóch list wraz z podaniem wyniku połączenia. Wśród danych programu mogą wystąpić klauzule:

  :- append([], Y, Y) (połączenie zbioru pustego i Y daje w wyniku Y)

  append([X|L],Y,[X|Z]) :- append(L,Y,Z) (rozszerzenie łączonego zbioru zawsze odpowiednio rozszerza wynik).

- **Zapytanie:** ?- append(A, B, [1,2])
- Odpowiedź:

  A=[] B=[1,2]

  A=[1] B=[2]

  A=[1,2] B=[]

8. System regułowy

System regułowy (produkcji) realizuje **wnioskowanie wprzód**.

Jest podstawą wielu systemów ekspertowych.

Typowy system regułowy (produkcji) składa się z:

- **pamięci roboczej** – zawiera pozytywne literały bez zmiennych (zdania atomowe);
- zbioru **reguł (produkcji)** – czyli wyrażeń postaci:

  $P_1 \wedge ... \wedge P_n \Rightarrow akcja_1 \wedge ... \wedge akcja_m$, gdzie

  $P_1,...,P_n$ są literałami (formułami atomowymi), a $akcja_1,...,$ $akcja_m$ są działaniami, które należy wykonać wtedy, jeśli wszystkie zdania P_i są prawdziwe (tzn. znajdują się w pamięci roboczej). W takiej sytuacji regułę nazywamy **realizowalną**. Przykłady akcji - dodanie lub usunięcie elementu z pamięci roboczej.

Cykl pracy systemu regułowego

Praca systemu regułowego składa się z ciągu **cykli**.

Każdy cykl składa się z **trzech faz**:

1. **faza dopasowania** – poszukiwanie realizowalnych reguł;
2. **faza wyboru** akcji – wybór reguł do wykonania i ustalenie ich kolejności;
3. **faza wykonania** – wykonanie wybranych reguł.

Zasadniczym elementem **fazy dopasowania** jest procedura **unifikacji**, ale w praktycznych systemach wymagana jest jej efektywna implementacja.

Przykładem efektywnego algorytmu dopasowywania jest tzw. **algorytm sieci czasu rzeczywistego RT** (tzw. RT-sieć) w OPS-5. Ogranicza on złożoność unifikacji dzięki jednokrotnemu wydzieleniu wspólnych zależności wielu reguł oraz pamiętaniu wcześniej wykonanych unifikacji.

Dopasowanie – algorytm „sieci RT”

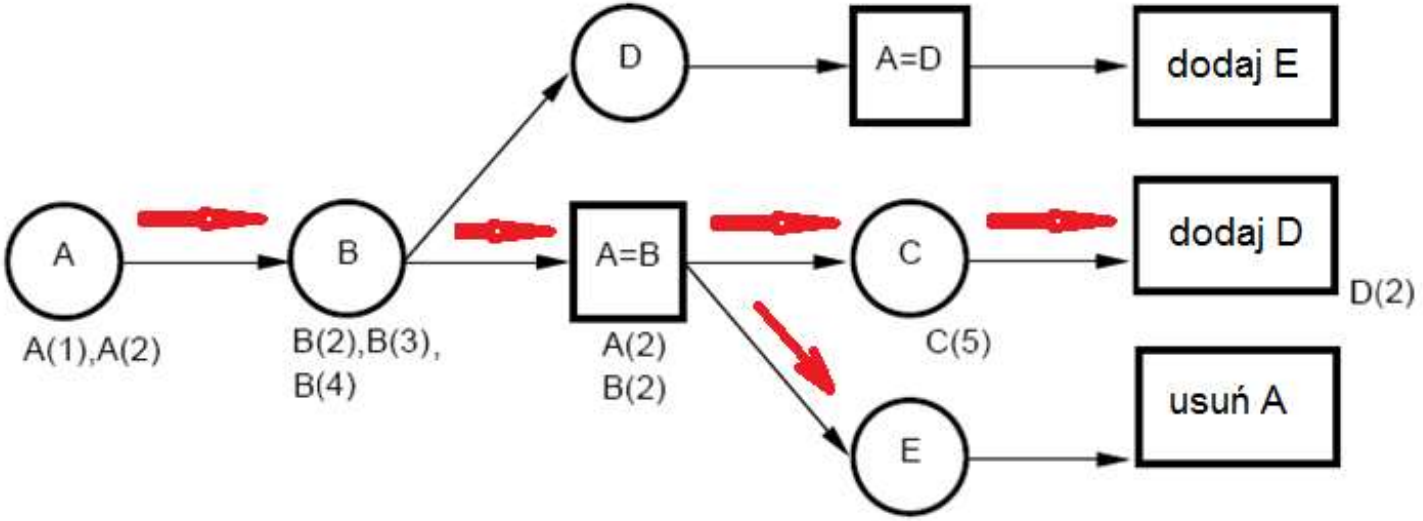
Przykład algortymu „sieć RT”

- Pamięć robocza: $\{ A(1), A(2), B(2), B(3), B(4), C(5) \}$
- Reguły: $A(x) \wedge B(x) \wedge C(y) \Rightarrow dodaj\ D(x).$

  $A(x) \wedge B(y) \wedge D(x) \Rightarrow dodaj\ E(x).$

  $A(x) \wedge B(x) \wedge E(z) \Rightarrow usuń\ A(x).$

Najpierw wykonywana jest konwersja reguł do sieci obliczeń (okręgi to odwołania do pamięci roboczej, kwadraty to unifikacje, prostokąty to wyprowadzane akcje).



Faza wyboru akcji

Faza wyboru – niektóre systemy wykonują wszystkie realizowalne działania, ale inne systemy wykonują tylko niektóre z nich. W takiej sytuacji są możliwe różne strategie wyboru:

Strategia (1) - preferujemy reguły, które odnoszą się do ostatnio tworzonych elementów pamięci roboczej;

Strategia (2) – preferujemy reguły bardziej „specyficzne”. Np. z poniższych reguł należy wybrać drugą:

$Ssak(x) \Rightarrow \text{add } Nogi(x,4).$

$Ssak(x) \wedge Człowiek(x) \Rightarrow \text{add } Nogi(x,2).$

Strategia (3) – preferujemy działania, którym nadano wyższy priorytet. Np. z poniższych reguł zapewne preferujemy drugą: $P(x) \Rightarrow Akcja(Odkurzanie(x)).$

$R(x) \Rightarrow Akcja(\text{Ewakuacja}).$

9. Sieci semantyczne (ramy)

Sieć semantyczna to reprezentacja wiedzy w postaci **grafu**, przedstawiającego **kategorie obiektów** (lub pojedyncze obiekty) i **relacje** pomiędzy nimi. Podstawowe **relacje** to:

- *Subset* – relacja pomiędzy kategorią ogólną a specjalizowaną (relacja dziedziczenia);
- *Part* – relacja części do całości;
- *Member* (lub Instance) – relacja instancji (obiektu) do jej kategorii.

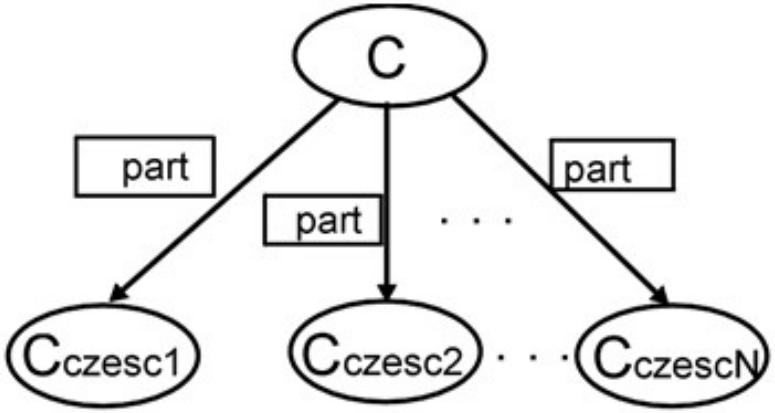
Każdy węzeł sieci (kategoria obiektów lub obiekt) posiada **nazwane atrybuty**, gdzie każdy atrybut wiąże obiekt z jakimś termem (w szczególności ze stałą). Atrybuty kategorii ogólnej są dziedziczone przez jej specjalizacje.

W zasadzie każda sieć semantyczna może być wyrażona w postaci zbioru formuł logiki 1 rzędu.

Sieć semantyczna a logika

- Zależność „zbiór części „part_of” → całość” jest interpretowana jako formuła implikacji:

  $C_{część1} \wedge C_{część2} \wedge ... \wedge C_{częśćN} \Rightarrow C$



- Podobnie, zależność „koncept_bazowy–spec-> koncept_pochodny” to implikacja postaci:

  $C_{pochodny} \Rightarrow C_{bazowy}$

Semantyka sieci semantycznej

Niech atrybut R dla węzła A wyznacza wartość B. **Semantyka atrybutu** - atrybut R może wyrażać relację 2-argumentową będącą 1 z 3 rodzajów:

- relacja zachodzi pomiędzy 2 obiektami: $R(A,B)$;
- relacja zachodzi pomiędzy każdym obiektem kategorii A i obiektem B: $\forall x \;\; x \in A \Rightarrow R(x,B)$;
- relacja zachodzi pomiędzy każdym obiektem kategorii A i pewnym elementem kategorii B: $\forall x \, \exists y \; x \in A \Rightarrow \mathrm{y} \in B \wedge R(x,y)$.

Jak wyrazić **wyjątki** dla relacji R zachodzące dla instancji kategorii A lub dla kategorii specjalizowanych?

- Odpowiednikiem atrybutu w logice *L1R* będzie formułą ze specjalnie wprowadzonym predykatem Rel : $Rel(R,A,B)$.

  Interpretacja $Rel(R,A,B)$: B jest wartością domyślną dla atrybutu R dla instancji kategorii A.

Pytania

1. Na czym polega **uogólniona reguła odrywania** („modus ponens”)?
2. Omówić procedurę wnioskowania „w przód” w logice predykatów.
3. Omówić procedurę wnioskowania „wstecz” w logice predykatów.
4. Na czym polega **reguła rezolucji** w logice predykatów?
5. Omówić elementy implementacji logicznego systemu wnioskowania.
6. Omówić techniki realizacji celu przez system Sztucznej Inteligencji.
7. Omówić system PROLOG.
8. Omówić system regułowy.
9. Przedstawić sieć semantyczną i jej związek z logiką predykatów.