**PROGRAMOWANIE W JĘZYKU LOGIKI – WPROWADZENIE**

**LOGIKA PIERWSZEGO RZĘDU**

Symbole języka pierwszego rzędu. dzielą się na:

a) symbole logiczne (wspólne dla wszystkich języków)

– zmienne przedmiotowe: $x$, $y$, $z$, …

– stałe logiczne: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists$

– symbole techniczne: ( , )

b) symbole pozalogiczne (zależne od języka)

– symbole relacyjne: P, Q, R, …

– symbole funkcyjne: f, g, h, …

– stałe przedmiotowe: a, b, c, …

(Symbole pozalogiczne całkowicie określają dany język)

*Językiem pierwszego rzędu* nazywamy układ

$$L = \left(\mathrm{Re}\,l_L; Fun_L; Con_L; \rho_L\right)$$

taki, że

– $\mathrm{Re}\,l_L$ jest niepustym zbiorem (symboli relacyjnych),

– $Fun_L$ jest zbiorem (symboli funkcyjnych),

– $Con_L$ jest zbiorem (stałych przedmiotowych),

przy czym zbiory $\mathrm{Re}\,l_L$, $Fun_L$ i $Con_L$ są rozłączne, natomiast $\rho_L$ jest funkcją, która każdemu symbolowi relacyjnemu i funkcyjnemu przyporządkowuje dodatnią liczbę całkowitą, zwaną arnością tego symbolu.

Wyróżniamy dwie klasy *wyrażeń sensownych* języka $L$:

a) *termy* – wyrażenia nazwowe,

b) *formuły* – wyrażenia zdaniowe.

*Termami* języka $L$ nazywamy wyrażenia języka $L$ określone przez następujące warunki indukcyjne:

– wszystkie zmienne i stałe przedmiotowe są termami ,

– $f(t_1, \ldots, t_n)$ jest termem, jeżeli $f \in Fun_L$, $\rho_L(f) = n$ oraz $t_1, \ldots, t_n$ są tremami.

*Formułami atomowymi* języka $L$ nazywamy wyrażenia

$$R(t_1, \ldots, t_n),$$

takie że $R \in \mathrm{Re}\, l_L$, $\rho_L(R) = n$ a $t_1, \ldots, t_n$ są tremami języka $L$.

*Formułami* języka $L$ nazywamy wyrażenia języka $L$ określone przez następujące warunki indukcyjne:

– wszystkie formuły atomowe są formułami,

– jeżeli $A$, $B$ są formułami, to wyrażenia $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \to B)$, $(A \leftrightarrow B)$ są formułami,

– jeżeli $A$ jest formułą i $x$ jest zmienną przedmiotową, to wyrażenia $(\forall x\ A)$ i $(\exists x\ A)$ są formułami.

**KLAUZULE**

*Literał pozytywny* – formuła atomowa (krótko: atom)

*Literał negatywny* – negacja atomu

*Literał* – literał pozytywny lub negatywny

*Klauzula* – formuła postaci $\forall(L_1 \vee L_2 \vee \ldots \vee L_n)$, gdzie $L_i$, $i = 1,2,\ldots,n$ są literałami.

Przykład klauzuli: $\forall x \forall y (P(x, y) \vee Q(f(x), h(y), a))$

Klauzula postaci:

$$\forall(A_1 \vee \ldots \vee A_k \vee \neg B_1 \vee \ldots \vee \neg B_m)$$

jest równoważna formule

$$\forall((A_1 \vee \ldots \vee A_k) \vee \neg(B_1 \wedge \ldots \wedge B_m)),$$

która z kolei jest równoważna formule

$$\forall(B_1 \wedge \ldots \wedge B_m \to A_1 \vee \ldots \vee A_k).$$

Formułę tę zapisujemy w postaci:

$$\underbrace{A_1,\ldots,A_k}_{wniosek} \leftarrow \underbrace{B_1,\ldots,B_m}_{przeslanka},$$

przecinki w przesłance (poprzednik implikacji) oznaczają koniunkcje,

przecinki we wniosku (następnik implikacji) oznaczają alternatywy.

**Przykład:**

klauzula: $\forall x \forall y (P(x) \vee \neg A \vee \neg Q(y) \vee B)$

zapis: $P(x), B \leftarrow A, Q(y)$

**Szczególne przypadki klauzul:**

$k = 1$

Wtedy klauzula jest postaci:

$$\boxed{A \leftarrow B_1, \ldots, B_m}$$

Klauzulę tej postaci nazywamy klauzula *definitywną*.

W szczególności $m = 0$.

Wtedy po prawej stronie otrzymujemy pustą koniunkcję (brak przesłanek).

Pusta koniunkcja jest zawsze prawdziwa.

Zatem w tym przypadku klauzula jest postaci:

$$\boxed{A \leftarrow \textbf{Prawda}}$$

Klauzulę taką nazywamy *jednostkową*.

$k = 0, m > 0$.

Wtedy po prawej stronie otrzymujemy pustą alternatywę (brak wniosków).

Pusta alternatywa jest zawsze fałszywa.

Zatem w tym przypadku klauzula jest postaci:

$$\boxed{\textbf{Fałsz} \leftarrow B_1, \ldots, B_m}$$

Klauzulę taką nazywamy *negatywną*.

$k = 0, m = 0$.

Wtedy zarówno koniunkcja jak i alternatywa są puste.

Mamy następującą sytuację :

$$\boxed{\textbf{Fałsz} \leftarrow \textbf{Prawda}}$$

Zatem w tym przypadku klauzula jest fałszywa.

Nazywamy ja klauzulą *pustą* i oznaczamy $\Box$

**Program w języku logiki.**

Programem w języku logiki nazywamy zbiór klauzul postaci:

| |
|---|
| $A \leftarrow B_1, \ldots, B_n$, $\quad n \geq 0$ |
| nagłówek treść, ciało |

Dla $n > 0$ klauzulę definitywną nazywamy regułą.
Dla $n = 0$ klauzula definitywna jest klauzula jednostkową i nazywamy ją faktem.

**Nieformalne znaczenie klauzuli definitywnej (reguły)**: dla każdego wartościowania zmiennych, jeżeli $B_i$, $i = 1,2,\ldots,m$ są prawdziwe, to $A$ jest prawdziwe.

**Nieformalne znaczenie klauzuli jednostkowej (faktu)**: $A$ jest prawdziwe dla każdego wartościowania zmiennych.

**Interpretacja** klauzul programu w języku logiki:

$$A \leftarrow B_1, \ldots, B_m, \qquad n \geq 0$$

a) deklaratywna (opisowa): $A$ jest prawdziwe, jeśli $B_1, \ldots, B_m$ są prawdziwe,
b) proceduralna (operacyjna): aby rozwiązać $A$ rozwiąż $B_1, \ldots, B_m$.

Zbiór wszystkich klauzul programu **P** w języku logiki, w których nagłówku występuje $P / n$ (predykat $P$ o $n$ argumentach) tworzy **definicję predykatu** (relacji, związku, własności) $P / n$. W istocie program **P** w języku logiki jest zbiorem definicji predykatów.

Program **P** w języku logiki jest opisem obiektów koniecznych do rozwiązania danego zagadnienia oraz związków jakie zachodzą pomiędzy tymi obiektami.

Program **P** w języku logiki jednoznacznie określa język pierwszego rzędu $L_P$.

Zadanie do rozwiązania przedstawione jest w postaci tzw. *celu* (ang. goal), *zapytania.*

*Zapytanie* jest klauzulą negatywna *N*, taką że $N \in L_P$.

*Wykonanie programu* polega na udowodnieniu, że podany cel wynika logicznie ze zbioru formuł tworzących program **P**.

W praktyce wykazujemy, że zbiór formuł $\mathbf{P} \cup \{N\}$ nie jest spełniany, co w programowaniu w logice sprowadza się do sprawdzenia, czy ze zbioru formuł $\mathbf{P} \cup \{N\}$ można wyprowadzić klauzulę pustą stosując *regułę rezolucji liniowej* (odpowiadającej stosowaniu bardziej klasycznych: reguły odrywania i reguły podstawiania).

Cel *N* ma postać:

$$\leftarrow B_1, \ldots, B_m,$$

czyli

$$\forall(B_1 \wedge \ldots \wedge B_m \rightarrow Falsz).$$

Stąd na podstawie prawa KRZ: $(p \rightarrow Falsz) \rightarrow \neg p$ mamy :

$$\forall(\neg(B_1 \wedge \ldots \wedge B_m)) \ ,$$

co jest równoważne

$$\neg\exists(B_1 \wedge \ldots \wedge B_m).$$

Jeżeli wykażemy, ze zbiór $\mathbf{P} \cup \{N\}$ nie jest spełnialny, to ze znanego faktu z logiki otrzymujemy, że formuła $\neg N$ wynika logicznie ze zbioru formuł tworzących program P.

$$\neg N \Leftrightarrow \neg\neg\exists(B_1 \wedge \ldots \wedge B_m) \Leftrightarrow \exists x_1 \ldots \exists x_k (B_1 \wedge \ldots \wedge B_m),$$

gdzie $x_1, \ldots, x_k$ są zmiennymi występującymi w *N*. Tym samym znajdziemy obiekty spełniające cel *N*.

**REGUŁA REZOLUCJI ZDANIOWEJ**

$$\frac{\begin{array}{c}\{L_1, L_2, \ldots, L_m, p\}\\ \{L_1', L_2', \ldots, L_n', \neg p\}\end{array}}{\{L_1, L_2, \ldots, L_m, L_1', L_2', \ldots, L_n'\}}, \quad n, m \geq 0.$$

**FAKT**
Reguła rezolucji zdaniowej jest logiczną regułą wnioskowania, tzn. wniosek reguły rezolucji zdaniowej wynika ze zbioru przesłanek tej reguły.

**REZOLUCJA LINIOWA.**

Niech P będzie programem definitywnym.

**Reguła rezolucji liniowej** ma postać:

$\leftarrow A_1, \ldots, A_m, \ldots, A_n$ – cel $G$

$B \leftarrow B_1, \ldots, B_k$ – wariant klauzuli $C$ programu $P$

---

$\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_k, A_{m+1}, \ldots, A_n)\sigma$ – nowy cel

gdzie $\sigma$ jest MGU zbioru $\{A_m, B\}$, tzn. $A_m\sigma = B\sigma$.

**Uzasadnienie:**

$G$: $\neg(A_1 \wedge \ldots \wedge A_m \wedge \ldots \wedge A_n) \Leftrightarrow \neg A_1 \vee \ldots \vee \neg A_m \vee \ldots \vee \neg A_n$
$C$: $B \vee \neg B_1 \vee \ldots \vee \neg B_k$
$\sigma$: $B\sigma = A_m\sigma$

$G\sigma$: $\neg A_1\sigma \vee \ldots \vee \neg A_m\sigma \vee \ldots \vee \neg A_n\sigma$
$C\sigma$: $B\sigma \vee \neg B_1\sigma \vee \ldots \vee \neg B_k\sigma$

rezolucja zdaniowa

---

$$\neg A_1\sigma \vee \ldots \vee \neg A_{m-1}\sigma \vee \neg B_1\sigma \vee \ldots \vee \neg B_k\sigma \vee \neg A_{m+1}\sigma \vee \ldots \vee \neg A_n\sigma$$
$$\Updownarrow$$
$$\neg(A_1 \wedge \ldots \wedge A_{m-1} \wedge B_1 \wedge \ldots \wedge B_k \wedge A_{m+1} \wedge \ldots \wedge A_n)\sigma$$