Zastosowanie metody rezolucji

1. Dedukcyjne bazy danych.
   Reprezentując jakąkolwiek rzeczywistość chcemy mieć możliwość otrzymywania odpowiedzi na różne pytania dotyczące tej rzeczywistości. Potraktowanie bazy wiedzy w sposób dedukcyjny pozwala uzyskać informacje, które w bazie wiedzy nie występują explicite.

2. Programowanie w logice.
   Metoda rezolucji pozwala nie tylko wykazać sprzeczność pewnego zbioru klauzul, ale jednocześnie znaleźć obiekty powodujące tę sprzeczność. Innymi słowy, dowody takie są konstruktywne. Fakt ten leży u podstaw języka programowania skonstruowanego na bazie metody rezolucji. Programowanie w takim języku ma charakter deklaratywny.

Dedukcyjne bazy danych

Rozważymy trzy klasy pytań:

A. Pytania typu “czy". Oczekiwaną odpowiedzią jest “tak” lub “nie". Np. Czy Jan jest żonaty?

B. Pytania typu “gdzie", “kto", “kiedy". Np. gdzie mieszka Piotr?, Kto kocha Marię?, Kiedy będę bogaty? Oczekiwane odpowiedzi: “w Warszawie", “Marek", “jak wygram na loterii".

C. Pytania typu “w jaki sposób". Oczekiwaną odpowiedzią jest ciąg działań prowadzących do osiągnięcia pewnego celu. Np. W jaki sposób można dojechać z Łodzi do Londynu? Odpowiedź: “Najpierw pociągiem z Łodzi do Warszawy, potem samolotem z Warszawy do Londynu".

Pytania klasy A

Zadanie otrzymania odpowiedzi sprowadza się do wykazania twierdzenia reprezentującego bezpośrednio dane pytanie.

Przykład 1

Rozważmy bazę wiedzy:

1. Wszystkie ptaki latają.
2. Wszystkie latające obiekty mają skrzydła.
3. Kuba jest ptakiem.

Pytanie:

Czy Kuba ma skrzydła?

Formalizacja

1. $\neg ptak(x) \vee lata(x)$
2. $\neg lata(x) \vee maskrzydla(x)$
3. $ptak(Kuba)$
4. $\neg ma - skrzydla(Kuba)$

Należy pokazać, że zbiór klauzul (1)- (4) jest niespełnialny.

Jeżeli formuła reprezentująca pytanie daje się wyprowadzić, odpowiedź jest “tak". W przeciwnym przypadku próbujemy wykazać jej zaprzeczenie. Jeśli uda się wyprowadzić zaprzeczenie, odpowiedź jest “nie". Jeśli się nie uda, odpowiedź jest “nie wiem”.

Pytania klasy B

Uzyskanie odpowiedzi jest teraz nieco bardziej skomplikowane.

Przykład 2

Rozważmy bazę wiedzy (1)- (3) oraz

Pytanie: Kto ma skrzydła?

Jaka formuła reprezentuje takie pytanie?

Stawiamy hipotezę:

$\exists x.ma - skrzydla(x)$

Po zanegowaniu i przekształceniu do postaci klauzulowej otrzymujemy

5. $\neg ma - skrzydla(x)$

Wykazujemy teraz niespełnialność zbioru klauzul (1)-(3),(5). W kolejnych krokach dowodu rezolucyjnego wykonujemy podstawienia $\{x'/x\}$ oraz $\{x/Kuba\}$. I właśnie $Kuba$ jest odpowiedzią na nasze pytanie.

Ogólnie:
Aby uzyskać odpowiedź na pytanie klasy B, konstruujemy formułę gwarantującą istnienie obiektu spełniającego warunki pytania, czyli formułę postaci $\exists x.F$. Jeśli formuła ta daje się dowieść metodą rezolucji i $\lambda_1, \ldots, \lambda_n$ są kolejnymi unifikatorami zastosowanymi podczas dowodu, to term $x\lambda_1 \circ \ldots \circ \lambda_n$ jest odpowiedzią.

Bardziej efektywną metodą śledzenia interesującej nas zmiennej jest wprowadzenie nowego symbolu predykatu, np. Odp. Zamiast formuły $\exists x.F$ będziemy próbowali wykazać formułę $\exists x.(F \wedge \neg Odp(x))$. Oczywiście teraz jako koniec dowodu dostajemy nie klauzulę pustą, ale klauzulę postaci $Odp(term)$, gdzie $term$ jest naszą odpowiedzią.

Wróćmy do naszego przykładu. Zamiast klauzuli (5) stosujemy klauzulę

6. $\neg ma - skrzydla(x) \vee Odp(x)$

W trakcie dowodu rezolucyjnego otrzymujemy klauzulę

7. $Odp(Kuba)$

która kończy dowód, a odpowiedzią jest $Kuba$.

Pytania klasy C

Agent reaktywny reaguje na bieżącą obserwację w oparciu o aktualny stan świata. Baza wiedzy takiego agenta musi więc mieć charakter dynamiczny.

Rachunek sytuacyjny (situation calculus, Hayes, McCarthy 1969) – system oparty na logice pierwszego rzędu do formalizacji dynamicznie zmieniającego się świata.

Świat = ciąg **sytuacji**, każda opisuje stan świata w pewnym momencie czasu. Nowa sytuacja powstaje z bieżącej sytuacji w wyniku wykonania akcji.

Każdy symbol reprezentujący relację zmieniającą się w czasie, posiada dodatkowy argument, odpowiadający sytuacji: np.

$$In(\textit{Łódź}, S_0) \wedge In(\textit{Warszawa}, S_1).$$

Do obsługi sytuacji wprowadzamy specjalny symbol funkcyjny *Result*

Result(action,situation) – sytuacja będąca wynikiem wykonania akcji *action* w sytuacji *situation*.

Definiowanie akcji

Akcje definiuje się podając ich efekty, np.

$$\forall z, do, s.\ In(do, \mathit{Result}(\mathit{Jedź}(z, do), s)).$$

Potrzebne są jeszcze **aksjomaty tła (frame axioms)**:

1. Jeśli fakt F zachodził przed wykonaniem akcji (tj. $F(S_0)$) i wykonana akcja go nie zmieniła, to F zachodzi w następnej sytuacji.
2. Jeśli fakt F nie zachodził przed wykonaniem akcji (tj. $\neg F(S_0)$) i nie jest jej efektem, to F nie zachodzi w następnym stanie.

$$\forall a, x, s\ In(x, s) \wedge (a \neq \mathit{Jedź}(x, y)) \Rightarrow In(x, \mathit{Result}(a, s)).$$

Programowanie w logice

Klauzula Horna – klauzula zawierająca co najwyżej jeden literał dodatni.

Każda klauzula Horna jest jednej z następujących postaci:

$$\neg P_1 \vee \cdots \vee \neg P_n \vee Q \qquad (n > 0) \qquad (1)$$

$$Q \qquad (2)$$

$$\neg P_1 \vee \cdots \vee \neg P_n \qquad (n > 0) \qquad (3)$$

$$\bot \qquad (4)$$

W programowaniu w logice klauzule (1)-(4) zapisujemy:

$Q \leftarrow P_1, \ldots P_n.$

$Q \leftarrow .$

$\leftarrow P_1, \ldots, P_n.$

$\bot.$

Klauzule Horna posiadają odpowiednią interpretację proceduralną.

- Klauzula postaci $Q \leftarrow P_1, \ldots P_n$ traktowana jest jako procedura o nagłówku Q i treści $P_1, \ldots P_n$. Wywołanie procedury Q sprowadza się do kolejnego wywołania procedur $P_1, \ldots P_n$.

- Klauzula postaci Q traktowana jest jako procedura o pustej treści.

- Klauzulę postaci $\leftarrow P_1, \ldots, P_n$, nazywaną **klauzulą celu**, można traktować jako dane programu.

- Klauzula pusta traktowana jest jako instrukcja *stop*.

Program – uporządkowany ciąg procedur.

Dane – klauzula celu.

Realizacja – (uproszczona) rezolucja liniowa.

Programowanie w logice jest programowaniem **deklaratywnym**: program jest logiczną specyfikacją problemu. Niestety języki programowania w logice nie są w pełni deklaratywne: w celu zwiększenia efektywności języki te dopuszczają wiele mechanizmów pozalogicznych.

Języki programowania w logice posiadają duży zbiór wbudowanych predykatów związanych z arytmetyką i wejściem/wyjściem. Literały odpowiadające tym predykatom realizowane są wykonaniem odpowiedniego kodu, a nie rezolucją.