Sztuczna inteligencja : reprezentacja i wyszukiwanie

(II / XVI)

ĆWICZENIA

1. Korzystając z tabel prawdy, udowodnij tożsamość w sekcji "Semantyka rachunku zdań".

2. Nowy operator ⊕ lub exlusive-or lub może zostać zdefiniowany w poniższej tabeli prawdy:

P	Q	P ⊕ Q
T	T	F
T	F	T
F	T	T
F	F	F

Utwórz wyrażenie rachunku zdań, używając tylko ∧, ∨, i ¬, które jest równoważne P ⊕ Q. Udowodnij ich równoważność za pomocą tabel prawdy

3. Operator logiczny "↔" jest odczytywany "wtedy i tylko wtedy, gdy". P↔Q jest zdefiniowany jako równoważny (P→Q) ∧ (Q→P). Na podstawie tej definicji pokaż, że P↔Q jest logicznie równoważne (P ∨ Q)→ (P ∧ Q):

a. Za pomocą tabel prawdy.

4. Udowodnij, że implikacja jest przechodnia w rachunku zdań, to znaczy, że ((P → Q) ∧ (Q → R)) → (P → R).

5. a. Udowodnij, że modus ponens jest poprawny dla rachunku zdań. Wskazówka: użyj tabel prawdy, aby wyliczyć wszystkie możliwe interpretacje.

b. Uprowadzenie to zasada wnioskowania, która wnioskuje P z P → Q i Q. Pokaż, że uprowadzenie nie jest dźwiękiem

c. Pokaż modus tollens ((P→Q) ∧ ¬ Q)→¬ P to dźwięk.

6. Spróbuj ujednolicić następujące pary wyrażeń. Albo pokaż ich najbardziej ogólne unifikatory lub wyjaśnij, dlaczego nie będą jednoczyć.

a. p(X,Y) i p(a,Z)

b. p(X,X) i p(a,b)

c. przodek(X Y) i przodek(rachunek, ojciec (rachunek))

d. przodek(X,ojciec(X)) i przodek (David,George)

e. q(X) i ¬ q (a)

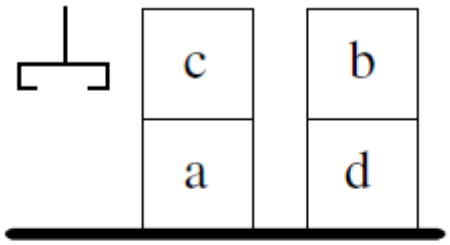
7. a. Skomponuj zestawy podstawień {a/X,Y/Z} i {X/W, b/Y}.

b. Udowodnij że skład zestawów podstawień jest asocjacyjny.

c. Skonstruuj przykład, aby pokazać, że kompozycja nie jest przemienna.

8. Zaimplementuj algorytm ujednolicenia opisany w części Unifikacja w wybranym języku komputerowym.

9. Podaj dwie alternatywne interpretacje opisu świata bloków na rysunku



on(c,a)
on(b,d)
ontable(a)
ontable(d)
clear(b)
clear(c)
hand_empty

10. Jane Doe ma cztery osoby na utrzymaniu, o stałym dochodzie w wysokości 30 000 USD i 15 000 USD na swoim koncie oszczędnościowym. Dodaj odpowiednie predykaty opisujące jej sytuację do ogólnego doradcy inwestycyjnego z przykładu "Aplikacja: Doradca finansowy oparty na logice" i wykonaj ujednolicenia i wnioski niezbędne do ustalenia jej sugerowanej inwestycji.

11. Napisz zestaw logicznych predykatów, które przeprowadzą prostą diagnostykę samochodową (np. Jeśli silnik się nie przekręci, a lampki nie zaświecą, oznacza to, że akumulator jest zły). Nie staraj się być zbyt skomplikowany, ale przykryj przypadki złej baterii, braku gazu, złych świec zapłonowych i złego silnika rozrusznika.

12. Poniższa historia pochodzi od N. Wirtha (1976) Algorytmy + struktury danych = programy. Poślubiłem wdowę (nazwijmy ją W), która ma dorosłą córkę (nazwij ją D). Mój ojciec (F), który często nas odwiedzał, zakochał się w mojej pasierbicy i poślubił ją. Dlatego mój ojciec został moim zięciem, a macocha stała się moją matką. Kilka miesięcy później moja żona urodziła syna (S1), który został szwagrem mojego ojca, a także wujkiem. Żona mojego ojca, czyli mojej przyrodniej córki, również miała syna (S2). Za pomocą rachunku predykatu utwórz zestaw wyrażeń, które reprezentują sytuację w powyższej historii. Dodaj wyrażenia określające podstawowe relacje rodzinne, takie jak definicja teścia i użyj modus ponens w tym systemie, aby udowodnić wniosek, że "jestem moim dziadkiem".

Sztuczna inteligencja : reprezentacja i wyszukiwanie(II)

Sztuczna inteligencja jako reprezentacja i poszukiwanie

Propozycja projektu badawczego Darthmouth Summer dotyczącego sztucznej inteligencji

„Proponujemy, aby latem 1956 r. W Dartmouth College w Hanover w stanie New Hampshire przeprowadzono dwumiesięczne, 10-osobowe badanie sztucznej inteligencji. Badanie ma przebiegać na podstawie przypuszczenia, że każdy aspekt uczenia się lub jakakolwiek inna cecha inteligencji może być w zasadzie tak precyzyjnie opisana, że można stworzyć maszynę do jej symulacji. Zostanie podjęta próba znalezienia sposobu zmuszenia maszyn do używania języka, tworzenia abstrakcji i pojęć, rozwiązywania rodzajów problemów zarezerwowanych obecnie dla ludzi i poprawy się. Uważamy, że można dokonać znaczącego postępu w zakresie jednego lub więcej z tych problemów, jeśli starannie wybrana grupa naukowców będzie pracować nad tym razem przez lato."

J. MCCARTHY, Dartmouth College , M. L. MINSKY, Harvard University , N. ROCHESTER, I.B.M. Corporation , C.E. SHANNON, Bell Telephone Laboratories , 31 sierpnia 1955 r

Wprowadzenie do reprezentacji i wyszukiwania

Z punktu widzenia inżynierii opis sztucznej inteligencji przedstawiony wcześniej można podsumować jako badanie reprezentacji i wyszukiwania, poprzez które inteligentna aktywność może być realizowana na mechanicznym urządzeniu. Ta perspektywa zdominowała początki i rozwój sztucznej inteligencji. Pierwsze nowoczesne warsztaty / konferencje dla praktyków AI odbyły się w Dartmouth College latem 1956 r. Propozycję tych warsztatów przedstawiono jako wstępny cytat z Części II. Warsztaty, na których wybrano samą nazwę sztuczna inteligencja, zgromadziły wielu ówczesnych badaczy skupionych na integracji obliczeń i inteligencji. W tym czasie napisano także kilka programów komputerowych odzwierciedlających te wczesne pomysły. Główne tematy do dyskusji na tej konferencji, skrócone tutaj od oryginalnej propozycji warsztatów, to:

1. Komputery automatyczne : Jeśli maszyna może wykonać zadanie, można zaprogramować automatyczny kalkulator do symulacji maszyny.

2. Jak można zaprogramować komputer do używania języka : Można spekulować, że duża część ludzkiej myśli polega na manipulowaniu słowami zgodnie z regułami rozumowania i przypuszczeniami.

3. Sieci neuronowe : Jak można zestaw (hipotetycznych) neuronów ułożyć koncepcje?

4. Teoria wielkości obliczenia : Jeśli otrzymamy dobrze zdefiniowany problem (taki, dla którego można mechanicznie sprawdzić, czy proponowana odpowiedź jest poprawną odpowiedzią), jednym ze sposobów rozwiązania tego problemu jest wypróbowanie wszystkich możliwych odpowiedzi w kolejności. Ta metoda jest nieefektywna i aby ją wykluczyć, trzeba mieć pewne kryterium wydajności obliczeń.

5. Samodoskonalenie (uczenie maszynowe) : Prawdopodobnie prawdziwie inteligentna maszyna będzie wykonywać czynności, które najlepiej można opisać jako samodoskonalenie.

6. Abstrakcje : Wiele rodzajów "abstrakcji" można wyraźnie zdefiniować, a kilka innych mniej wyraźnie. Bezpośrednia próba ich sklasyfikowania i opisania maszynowych metod formowania abstrakcji na podstawie danych sensorycznych i innych wydaje się opłacalna.

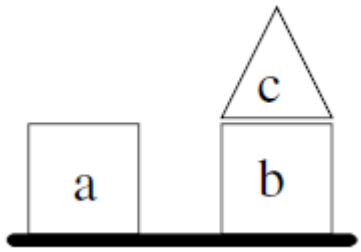
7. Losowość i kreatywność : Dość atrakcyjną, a jednak wyraźnie niekompletną hipotezą jest to, że różnica między kreatywnym myśleniem a niewyobrażalnym, kompetentnym myśleniem polega na zastrzyku pewnej przypadkowości.

Warto zauważyć, że tematy zaproponowane na pierwszą konferencję na temat sztucznej inteligencji obejmują wiele zagadnień, takich jak teoria złożoności, metodologia abstrakcji, projektowanie języków i uczenie maszynowe, które stanowią przedmiot współczesnej informatyki. W rzeczywistości wiele charakterystycznych cech informatyki, jaką znamy dzisiaj, ma swoje korzenie w sztucznej inteligencji. Sztuczna inteligencja również miała swoje historyczne i polityczne zmagania, a kilka z tych wczesnych tematów zaproponowanych do badań, takich jak "sieci neuronowe" i "losowość i kreatywność" zostały wprowadzone w tryb tła na dziesięciolecia. Mniej więcej w tym czasie pojawiło się nowe, potężne narzędzie obliczeniowe, język Lisp, zbudowane pod kierunkiem Johna McCarthy'ego, jednego z pierwszych twórców Warsztatu Dartmouth. Lisp zajął się kilkoma tematami Warsztatu, wspierając zdolność do tworzenia relacji, którymi mogłyby manipulować inne struktury języka. Lisp dał sztucznej inteligencji zarówno bardzo wyrazisty język, bogaty w abstrakcję, jak i medium do interpretacji tych wyrażeń. Dostępność języka programowania Lisp ukształtowała znaczną część wczesnego rozwoju sztucznej inteligencji, w szczególności wykorzystanie rachunku predykatów jako medium reprezentacyjnego, a także poszukiwanie w celu zbadania skuteczności różnych logicznych alternatyw, które obecnie nazywamy wyszukiwaniem grafowym. Prolog, stworzony pod koniec lat siedemdziesiątych, oferował AI podobne potężne narzędzie obliczeniowe.

Systemy reprezentacyjne

Funkcją każdego schematu reprezentacji jest uchwycenie - często nazywane streszczeniem - kluczowych cech domeny problemowej i udostępnienie tych informacji procedurze rozwiązywania problemów. Abstrakcja jest niezbędnym narzędziem do zarządzania złożonością, a także ważnym czynnikiem zapewniającym, że uzyskane programy są wydajne obliczeniowo. Ekspresyjność (wynik abstrakcyjnych cech) i wydajność (złożoność obliczeniowa algorytmów zastosowanych w abstrakcyjnych cechach) są głównymi wymiarami do oceny języków reprezentacji wiedzy. Czasami ekspresja musi zostać poświęcona, aby poprawić wydajność algorytmu. Należy tego dokonać bez ograniczania zdolności reprezentacji do przechwytywania niezbędnej wiedzy na temat rozwiązywania problemów. Optymalizacja kompromisu między wydajnością a ekspresją jest głównym zadaniem projektantów inteligentnych programów. Języki reprezentacji wiedzy mogą być również narzędziami pomagającymi ludziom w rozwiązywaniu problemów. Jako taka reprezentacja powinna zapewniać naturalne ramy dla wyrażania wiedzy na temat rozwiązywania problemów; powinna udostępnić tę wiedzę komputerowi i pomóc programiście w organizacji. Komputerowa reprezentacja liczb zmiennoprzecinkowych ilustruje te kompromisy. Mówiąc ściślej, liczby rzeczywiste wymagają pełnego opisu nieskończonego ciągu cyfr; nie można tego zrobić na urządzeniu skończonym, takim jak komputer. Jedną z odpowiedzi na ten dylemat jest przedstawienie liczby w dwóch częściach: cyfr znaczących i położenia w tych cyfrach miejsca dziesiętnego. Chociaż nie można faktycznie zapisać liczby rzeczywistej w komputerze, możliwe jest utworzenie reprezentacji, która będzie działać poprawnie w większości praktycznych zastosowań. Reprezentacja zmiennoprzecinkowa poświęca zatem pełną moc ekspresji, aby reprezentacja była wydajna, w tym przypadku, aby była możliwa. Ta reprezentacja obsługuje również algorytmy arytmetyki wielokrotnej precyzji, zapewniając efektywnie nieskończoną precyzję poprzez ograniczenie błędu zaokrąglenia do dowolnej z góry określonej tolerancji. Gwarantuje również dobrze zachowane błędy zaokrąglania. Podobnie jak wszystkie reprezentacje, jest to tylko abstrakcja, wzór symbolu, który określa pożądany byt, a nie sam byt. Tablica to kolejna reprezentacja powszechna w informatyce. W przypadku wielu problemów jest bardziej naturalny i wydajny niż architektura pamięci zaimplementowana w sprzęcie komputerowym. Ten wzrost naturalności i

wydajności pociąga za sobą kompromisy w ekspresji. Scena wizualna składa się z kilku punktów obrazu. Każdy punkt obrazu lub piksel ma zarówno lokalizację, jak i wartość liczbową reprezentującą jego intensywność lub poziom szarości. Naturalne jest zatem zebranie całej sceny w dwuwymiarową tablicę, w której adres wiersza i kolumny podaje lokalizację piksela (współrzędne X i Y), a zawartość elementu tablicy jest w tym miejscu poziomem szarości. Algorytmy są zaprojektowane do wykonywania takich operacji, jak szukanie izolowanych punktów w celu usunięcia szumów z obrazu, znajdowanie poziomów progowych dla rozpoznawania obiektów i krawędzi, sumowanie sąsiadujących elementów w celu określenia rozmiaru lub gęstości oraz na różne inne sposoby przekształcanie danych punktów obrazu. Wdrażane ich algorytmy są proste, biorąc pod uwagę na przykład reprezentację tablic i język FORTRAN. To zadanie byłoby dość kłopotliwe przy użyciu innych reprezentacji, takich jak rachunek predykatów, rekordy lub kod asemblacji, ponieważ nie mają one naturalnego dopasowania do reprezentowanego materiału. Reprezentując obraz jako układ pikseli, poświęcamy dokładność rozdzielczości (porównaj zdjęcie w gazecie z oryginalnym drukiem tego samego obrazu). Ponadto tablice pikseli nie mogą wyrazić głębszej semantycznej organizacji obrazu. Na przykład macierz pikseli nie może reprezentować organizacji chromosomów w jądrze pojedynczej komórki, ich funkcji genetycznej ani roli metafazy w podziale komórek. Wiedza ta jest łatwiejsza do zdobycia przy użyciu takich reprezentacji, jak rachunek predykatów lub sieci semantyczne. Podsumowując, schemat reprezentatywny powinien być odpowiedni do wyrażenia wszystkich niezbędnych informacji, wspierania wydajnego wykonania wynikowego kodu i zapewnienia naturalnego schematu wyrażania wymaganej wiedzy. Zasadniczo problemy, które AI próbuje rozwiązać, nie nadają się do reprezentacji oferowanych przez bardziej tradycyjne formalizmy, takie jak tablice. Sztuczna inteligencja dotyczy raczej jakościowego niż ilościowego rozwiązywania problemów, rozumowania zamiast obliczeń numerycznych oraz organizowania dużej i zróżnicowanej ilości wiedzy zamiast wdrażania jednego, dobrze zdefiniowanego algorytmu. Na przykład rozważmy rysunek, układ bloków na stole.



Załóżmy, że chcemy uchwycić właściwości i relacje wymagane do sterowania ramieniem robota. Musimy ustalić, które bloki są ułożone w stos na innych blokach, a które mają przezroczyste wierzchołki, aby można je było podnieść. Rachunek predykatów oferuje medium do przechwytywania tych opisowych informacji. Pierwsze słowo każdego wyrażenia (on, ontable itp.) Jest predykatem oznaczającym pewną właściwość lub związek między jego argumentami (występującymi w nawiasach). Argumenty to symbole oznaczające obiekty (bloki) w domenie. Zbiór klauzul logicznych opisuje ważne właściwości i relacje tego świata bloków:

clear(c)

clear(a)

ontable(a)

ontable(b)

on(c, b)

cube(b)

cube(a)

pyramid(c)

Rachunek predykatów zapewnia programistom sztucznej inteligencji dobrze zdefiniowany język do opisywania i wnioskowania o jakościowych aspektach systemu. Załóżmy, że w przykładzie świata bloków chcemy zdefiniować test w celu ustalenia, czy blok jest czysty, czyli nie ma na nim niczego ułożonego na sobie. Jest to ważne, jeśli ręka robota ma ją podnieść lub ułożyć na niej kolejny blok. Możemy zdefiniować ogólną zasadę:

$$\forall X \neg \exists Y \text{ on}(Y,X) \Rightarrow \text{clear}(X)$$

Jest to odczytywane "dla wszystkich X, X jest jasne, jeśli nie ma Y takiego, że Y jest na X". Tę ogólną zasadę można zastosować w różnych sytuacjach, zastępując różne nazwy bloków, a, b, c itp. , dla X i Y. Wspierając ogólne reguły wnioskowania, rachunek predykatów pozwala na oszczędność reprezentacji, a także na możliwość zaprojektowania systemów, które są wystarczająco elastyczne i ogólne, aby inteligentnie reagować na szereg sytuacji. Rachunek predykatów można również wykorzystać do przedstawienia właściwości jednostek i grup. Często nie wystarczy na przykład opisać samochodu, po prostu wymieniając jego części; możemy chcieć opisać sposoby łączenia tych części i interakcje między nimi. Ten widok struktury jest niezbędny w szeregu sytuacji, w tym w informacjach taksonomicznych, takich jak klasyfikacja roślin według rodzaju i gatunku lub opis złożonych obiektów, takich jak silnik Diesla lub ciało ludzkie, pod względem ich części składowych. Na przykład prosty opis niebieskiego ptaka może brzmieć: "niebieski ptak to mały niebieski ptak, a ptak to pierzasty latający kręgowiec", który można przedstawić jako zbiór logicznych predykatów:

hassize(bluebird,small)

hascovering(bird,feathers)

hascolor(bluebird,blue)

hasproperty(bird,flies)

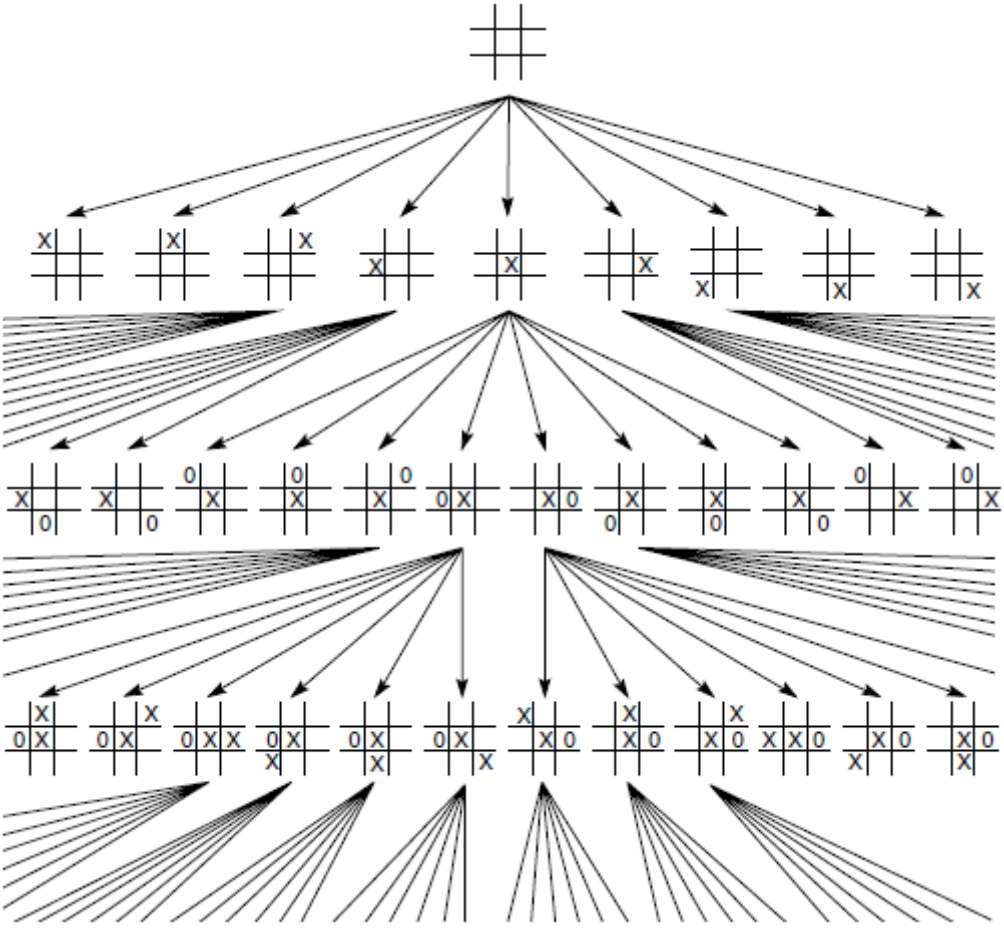
isa(bluebird,bird)

isa(bird,vertebrate)

Ten opis predykatu można przedstawić graficznie za pomocą łuków lub łączy na wykresie zamiast predykatów w celu wskazania relacji. Ta sieć semantyczna jest techniką reprezentowania znaczenia semantycznego. Ponieważ relacje są wyraźnie zaznaczone na wykresie, algorytm do wnioskowania o domenie problemowej może tworzyć odpowiednie powiązania, podążając za linkami. Na przykład na ilustracji bluebird, program musi podążać tylko za jednym linkiem, aby zobaczyć, że mucha leci, i dwoma linkami, aby ustalić, że bluebird jest kręgowcem. Być może najważniejszą aplikacją dla sieci semantycznych jest reprezentowanie znaczenia programów do rozumienia języka. Gdy konieczne jest zrozumienie historii dziecka, szczegółów artykułu w czasopiśmie lub zawartości strony internetowej, sieci semantyczne można wykorzystać do kodowania informacji i relacji odzwierciedlających wiedzę w tej aplikacji.

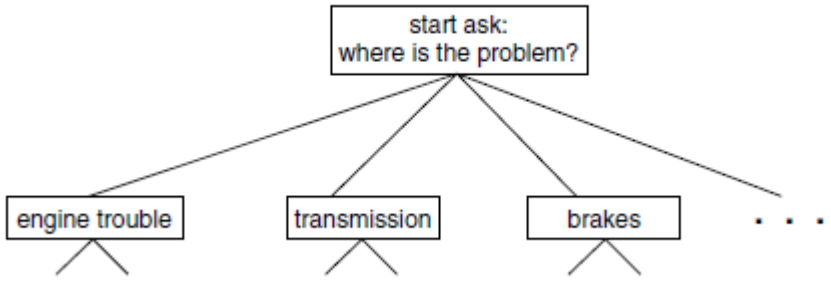
Wyszukiwanie

Biorąc pod uwagę reprezentację, drugim elementem inteligentnego rozwiązywania problemów jest wyszukiwanie. Ludzie na ogół rozważają szereg alternatywnych strategii na drodze do rozwiązania problemu. Gracz w szachy zazwyczaj sprawdza alternatywne ruchy, wybierając "najlepsze" zgodnie z kryteriami, takimi jak możliwe reakcje przeciwnika lub stopień, w jakim różne ruchy wspierają niektóre globalne strategie gry. Gracz bierze również pod uwagę krótkoterminowe korzyści (takie jak zabranie królowej przeciwnika), możliwości poświęcenia elementu dla uzyskania przewagi pozycyjnej lub domysły dotyczące psychologicznego wyglądu przeciwnika i poziomu umiejętności. Ten aspekt inteligentnego zachowania leży u podstaw techniki rozwiązywania problemów wyszukiwania w przestrzeni stanów. Rozważmy na przykład grę w kółko i krzyżyk. W każdej sytuacji na planszy gracz może wykonać tylko skończoną liczbę ruchów. Zaczynając od pustej planszy, pierwszy gracz może umieścić X w dowolnym z dziewięciu miejsc. Każdy z tych ruchów daje inną planszę, która pozwoli przeciwnikowi na osiem możliwych reakcji i tak dalej. Możemy przedstawić tę kolekcję możliwych ruchów i odpowiedzi, traktując każdą konfigurację płytki jako węzeł lub stan na wykresie. Łącza na wykresie przedstawiają legalne ruchy z jednej konfiguracji płyty na drugą. Powstała struktura jest wykresem w przestrzeni stanów. Reprezentacja przestrzeni stanu pozwala zatem traktować wszystkie możliwe gry w kółko i krzyżyk jako różne ścieżki na wykresie przestrzeni stanu. Biorąc pod uwagę tę reprezentację, skuteczna strategia gry przeszuka na wykresie ścieżki, które prowadzą do największej liczby wygranych i najmniejszych strat, i gra w sposób, który zawsze stara się wymusić grę wzdłuż jednej z tych optymalnych ścieżek, jak na rysunku

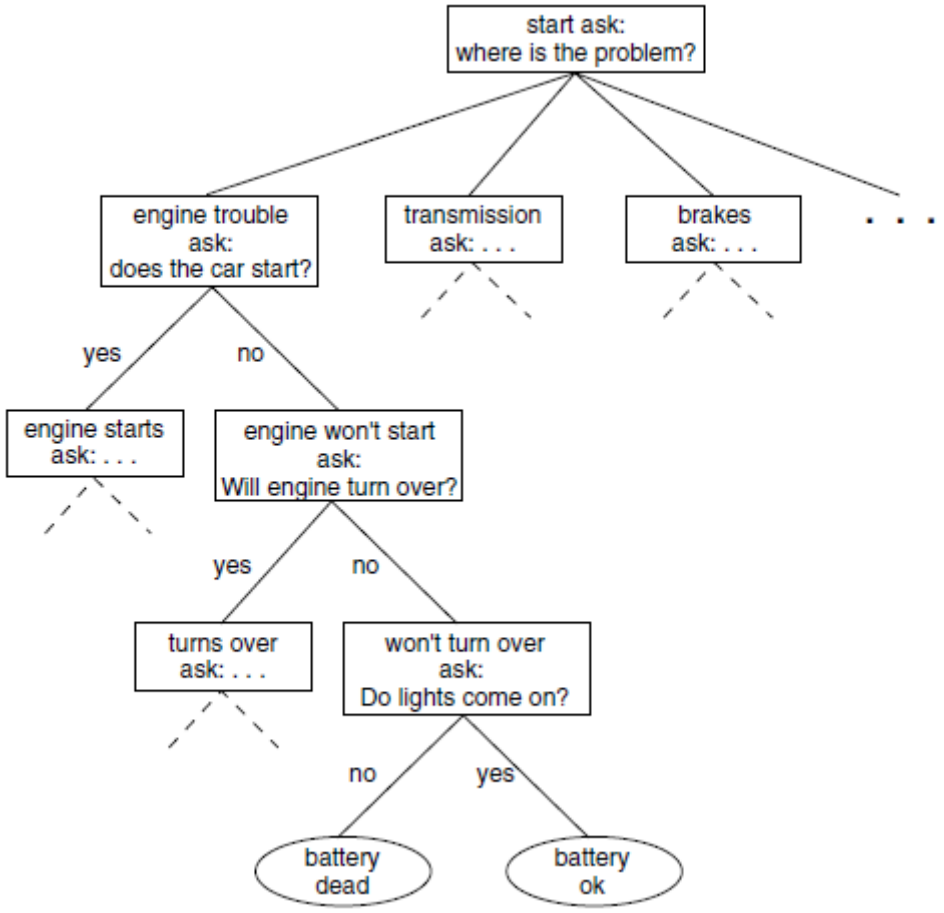


Jako przykład zastosowania wyszukiwania do rozwiązania bardziej skomplikowanego problemu, rozważ zadanie zdiagnozowania usterki mechanicznej w samochodzie. Chociaż początkowo problem ten nie wydaje się tak łatwy do przeszukiwania przestrzeni stanów jak kółko i krzyżyk lub szachy, w

rzeczywistości całkiem dobrze pasuje do tej strategii. Zamiast pozwolić, aby każdy węzeł wykresu reprezentował "stan tablicy", pozwalamy mu reprezentować stan częściowej wiedzy na temat problemów mechanicznych samochodu. Proces badania symptomów usterki i indukowania jej przyczyny można uznać za przeszukiwanie stanów o rosnącej wiedzy. Początkowy węzeł wykresu jest pusty, co wskazuje, że nic nie wiadomo na temat przyczyny problemu. Pierwszą rzeczą, jaką może zrobić mechanik, jest zapytanie klienta, który główny układ (silnik, skrzynia biegów, układ kierowniczy, hamulce itp.) Wydaje się powodować problemy. Jest to reprezentowane przez zbiór łuków od stanu początkowego do stanów, które wskazują na skupienie się na pojedynczym podsystemie samochodu, jak na rysunku



Na przykład węzeł usterki silnika ma łuki do węzłów oznaczone jako start silnika, a silnik nie chce się uruchomić. Z węzła "nie startuj" możemy przejść do węzłów oznaczonych jako zwroty i nie można go odwrócić. Węzeł, który nie przeszedł, ma łuki do węzłów oznaczone jako zużyta bateria i bateria w porządku



Narzędzie do rozwiązywania problemów może zdiagnozować awarię samochodu, szukając ścieżki na tym wykresie, która jest zgodna z objawami konkretnego uszkodzonego samochodu. Chociaż ten problem różni się bardzo od znalezienia optymalnego sposobu gry w kółko i krzyżyk lub w szachy, równie dobrze można go rozwiązać za pomocą wyszukiwania w przestrzeni stanów. Pomimo tej pozornej uniwersalności poszukiwanie przestrzeni stanów samo w sobie nie wystarcza do automatyzacji inteligentnego zachowania w rozwiązywaniu problemów; jest raczej ważnym narzędziem do projektowania inteligentnych programów. Gdyby przeszukiwanie przestrzeni stanu było wystarczające, napisanie programu, który gra w szachy, byłoby dość proste, przeszukując całą przestrzeń pod kątem sekwencji ruchów, które przyniosły zwycięstwo, metodą znaną jako wyczerpujące przeszukiwanie. Chociaż wyczerpujące wyszukiwanie można zastosować do dowolnej przestrzeni stanów, przytłaczająca wielkość przestrzeni dla interesujących problemów sprawia, że takie podejście jest praktycznie niemożliwe. Na przykład szachy mają około 10^{120} różnych stanów planszy. Jest to liczba większa niż liczba cząsteczek we wszechświecie lub liczba nanosekund, które minęły od Wielkiego Wybuchu. Poszukiwanie tej przestrzeni wykracza poza możliwości dowolnego urządzenia komputerowego, którego wymiary muszą być ograniczone do znanego wszechświata i którego wykonanie musi zostać zakończone, zanim wszechświat ulegnie spustoszeniu entropii. Ludzie używają inteligentnego wyszukiwania: szachista rozważa kilka możliwych ruchów, lekarz bada kilka możliwych diagnoz, informatyk bada różne projekty przed rozpoczęciem pisania kodu. Ludzie nie używają wyczerpującego wyszukiwania: szachista bada tylko ruchy, które doświadczalnie okazało się skuteczne, lekarz nie wymaga testów, które nie są w jakiś sposób wskazane przez objawy. Wydaje się, że ludzkie rozwiązywanie problemów opiera się na regułach oceny, które kierują wyszukiwaniem do tych części

przestrzeni państwowej, które wydają się najbardziej "obiecujące". Reguły te są znane jako heurystyka i stanowią jeden z głównych tematów badań nad AI. Heurystyka (nazwa pochodzi od greckiego słowa "odkryć") to strategia selektywnego przeszukiwania przestrzeni problemowej. Prowadzi wyszukiwanie wzdłuż linii, które mają duże prawdopodobieństwo sukcesu, unikając zmarnowanych lub pozornie głupich wysiłków. Ludzie używają dużej liczby heurystyk do rozwiązywania problemów. Jeśli zapytasz mechanika, dlaczego twój samochód się przegrzewa, może powiedzieć coś takiego: "Zwykle oznacza to, że termostat jest zły". Jeśli zapytasz lekarza, co może powodować nudności i bóle brzucha, może powiedzieć, że "prawdopodobnie jest to grypa żołądkowa lub zatrucie pokarmowe. "Wyszukiwanie w przestrzeni stanu pozwala nam sformalizować proces rozwiązywania problemów, a heurystyka pozwala nam nadać temu formalizmowi inteligencję. Techniki te pozostają w centrum najnowocześniejszych prac nad sztuczną inteligencją. Podsumowując, wyszukiwanie w przestrzeni stanu jest formalizmem, niezależnym od konkretnych strategii wyszukiwania i wykorzystywanym jako punkt wyjścia dla wielu podejść do rozwiązywania problemów. W całym tekście kontynuujemy eksplorację teoretycznych aspektów reprezentacji wiedzy i wyszukiwania oraz zastosowania tej teorii w budowaniu skutecznych programów.

Rachunek predykatów

Rachunek zdań

Symbole i zdania

Rachunek zdań i rachunek predykatów, są przede wszystkim językami. Używając ich słów, zwrotów i zdań, możemy reprezentować i uzasadniać właściwości i relacje na świecie. Pierwszym krokiem w opisie języka jest przedstawienie tworzących go elementów: zestawu symboli.

DEFINICJA

Symbole rachunku zdań

Symbole rachunku zdań są symbolami zdań:

P, Q, R, S, …

symbole prawdy: true i false

i łączniki: ∧, ∨, ¬, →, ≡

Symbole zdaniowe oznaczają zdania lub oświadczenia o świecie, które mogą być prawdziwe lub fałszywe, takie jak „samochód jest czerwony” lub „woda jest mokra”. Propozycje są oznaczone dużymi literami na końcu alfabetu angielskiego. Zdania w rachunku zdań są tworzone z tych symboli atomowych zgodnie z następującymi zasadami:

DEFINICJA

Zdania z rachunku zdań

Każdy symbol zdania i symbol prawdy to zdanie.

Na przykład: true, P, Q i R są zdaniami.

Negacja zdania jest zdaniem.

Na przykład: ¬ P i ¬ fałsz to zdania.

Koniunkcja lub and dwóch zdań jest zdaniem.

Na przykład: P ∧ ¬ P jest zdaniem.

Rozłączenie lub lub dwóch zdań jest zdaniem.

Na przykład: P ∨ ¬ P jest zdaniem.

Implikacja jednego zdania z drugiego jest zdaniem.

Na przykład: P → Q to zdanie.

Równoważność dwóch zdań jest zdaniem.

Na przykład: P ∨ Q ≡ R to zdanie.

Zdania legalne są również nazywane dobrze sformułowanymi formułami lub WFF.

W wyrażeniach postaci P ∧ Q, P i Q nazywane są połączonymi. W P ∨ Q, P i Q są nazywane rozłącznymi. W implikacji P → Q, P jest przesłanką lub poprzednikiem, a Q wnioskiem lub konsekwencją. W zdaniach zdań rachunku różniczkowego symbole () i [] są używane do grupowania symboli w podwyrażenia, a więc do kontrolowania ich kolejności oceny i znaczenia. Na przykład (P ∨ Q) ≡ R różni się znacznie od P ∨ (Q ≡ R), co można wykazać za pomocą tabel prawdy. Wyrażenie jest zdaniem lub dobrze sformułowaną formułą rachunku zdań, jeśli i tylko wtedy, gdy można je utworzyć z legalnych symboli poprzez pewną sekwencję tych reguł. Na przykład, ((P∧ Q) → R) ≡¬ P∨ ¬ Q∨ R jest dobrze uformowanym zdaniem w rachunku zdań, ponieważ:

P, Q i R to twierdzenia, a zatem zdania.

P ∧ Q, połączenie dwóch zdań, jest zdaniem.

(P ∧ Q) → R, implikacja zdania dla innego, jest zdaniem.

¬ P i ¬ Q, negacje zdań, są zdaniami.

¬ P ∨ ¬ Q, rozłączenie dwóch zdań, jest zdaniem.

¬ P ∨ ¬ Q ∨ R, rozłączenie dwóch zdań, jest zdaniem.

((P ∧ Q) → R) ≡ ¬ P ∨ ¬ Q ∨ R, równoważność dwóch zdań, jest zdaniem.

To jest nasze oryginalne zdanie, które zostało skonstruowane poprzez szereg zastosowań przepisów prawnych i dlatego jest „dobrze uformowane".

Semantyka rachunku zdań

We wcześniejszej sekcji przedstawiono składnię rachunku zdań, definiując zbiór zasad dotyczących wydawania wyroków sądowych. W tej sekcji formalnie definiujemy semantykę lub „znaczenie" tych zdań. Ponieważ programy AI muszą rozumować za pomocą swoich struktur reprezentacyjnych, ważne jest wykazanie, że prawdziwość ich wniosków zależy tylko od prawdy ich początkowej wiedzy lub przesłanek, tj. że błędy logiczne nie są wprowadzane przez procedury wnioskowania. Precyzyjne traktowanie semantyki jest niezbędne do osiągnięcia tego celu. Symbol propozycji odpowiada stwierdzeniu o świecie. Na przykład P może oznaczać stwierdzenie „pada deszcz" lub Q, stwierdzenie „Mieszkam w brązowym domu". Twierdzenie musi być prawdziwe lub fałszywe, biorąc pod uwagę pewien stan świata. Przypisanie wartości prawdy do zdań zdań jest nazywane interpretacją, twierdzeniem o ich prawdzie w jakimś możliwym świecie. Formalnie interpretacja jest odwzorowaniem symboli zdań na zbiór {T, F}. Jak wspomniano w poprzedniej sekcji, symbole prawda i fałsz są częścią zestawu dobrze uformowanych zdań rachunku zdań; tzn. różnią się od wartości prawdy przypisanej do

zdania. Aby wprowadzić to rozróżnienie, symbole T i F służą do przypisania wartości prawdy. Każde możliwe mapowanie wartości prawdy na zdania odpowiada możliwemu światowi interpretacji. Na przykład, jeśli P oznacza zdanie „pada deszcz", a Q oznacza „Jestem w pracy", to zestaw zdań {P, Q} ma cztery różne mapowania funkcjonalne na wartości prawdy {T, F}. Te odwzorowania odpowiadają czterem różnym interpretacjom. Semantyka rachunku zdań, podobnie jak jego składnia, jest definiowana indukcyjnie:

DEFINICJA

Semantyka rachunku zdań

Interpretacja zbioru zdań jest przypisaniem wartości prawdy, T lub F, do każdego symbolu zdania.

Symbolowi true przypisuje się zawsze T, a symbolowi false - F.

Interpretacja lub wartość prawdy dla zdań jest określona przez:

Prawidłowe przypisanie negacji, ¬P, gdzie P jest dowolnym symbolem zdaniowym, to F, jeśli przypisanie do P jest T, a T, jeśli przypisanie do P jest F.

Prawidłowe przypisanie koniunkcji ∧ jest T tylko wtedy, gdy obie koniunkcje mają wartość prawdy T; w przeciwnym razie jest to F.

Prawidłowe przypisanie rozłączenia ∨ jest F tylko wtedy, gdy oba rozłączenia mają wartość prawdy F; w przeciwnym razie jest to T.

Prawidłowe przypisanie implikacji, →, ma wartość F tylko wtedy, gdy przesłanką lub symbolem przed implikacją jest T, a wartość prawdy wyniku lub symbolu po implikacji wynosi F; w przeciwnym razie jest to T.

Przypisanie równoważności przez prawdę ≡ jest T tylko wtedy, gdy oba wyrażenia mają to samo przypisanie prawdy dla wszystkich możliwych interpretacji; w przeciwnym razie jest to F.

Przypisania prawdy zdań złożonych są często opisywane w tabelach prawdy. Tabela prawdy zawiera listę wszystkich możliwych przypisań wartości prawdy do zdań atomowych wyrażenia i podaje wartość prawdy wyrażenia dla każdego zadania. Zatem tabela prawdy wylicza wszystkie możliwe światy interpretacji, które można nadać wyrażeniu. Na przykład tabela prawdy dla P ∧ Q, zawiera wartości prawdy dla każdego możliwego przypisania prawdy operandów. P ∧ Q jest prawdziwe tylko wtedy, gdy P i Q są jednocześnie T. Lub (∨), nie (¬), implikuje (→) i równoważność (≡) są zdefiniowane w podobny sposób. Konstrukcja tabel prawdy pozostawia się jako ćwiczenie. Dwa wyrażenia w rachunku zdań są równoważne, jeśli mają tę samą wartość w ramach wszystkich przypisań wartości prawdy. Tę równoważność można wykazać za pomocą tabel prawdy. Na przykład dowód równoważności P → Q i ¬ P ∨ Q podano w tabeli prawdy na rycinie 2.2. Wykazując, że dwa różne zdania w rachunku zdań mają identyczne tabele prawdy, możemy udowodnić następujące równoważności. W przypadku wyrażeń zdań P, Q i R:

¬ (¬ P) ≡ P

(P ∨ Q) ≡ (¬ P → Q)

prawo przeciwstawności : (P → Q) ≡ (¬ Q → ¬P)

prawo de Morgana: ¬ (P ∨ Q) ≡ (¬ P ∧ ¬ Q) i ¬ (P ∧ Q) ≡ (¬ P ∨¬ Q)

prawa przemienne: (P ∧ Q) ≡ (Q ∧ P) i (P ∨ Q) ≡ (Q ∨ P)

prawo łączności: ((P ∧ Q) ∧ R) ≡ (P ∧ (Q ∧ R))

prawo łączności: ((P ∨ Q) ∨ R) ≡ (P ∨ (Q ∨ R))

prawo rozdzielności: P ∨ (Q ∧ R) ≡ (P ∨ Q) ∧ (P ∨ R)

prawo rozdzielności: P ∧ (Q ∨ R) ≡ (P ∧ Q) ∨ (P ∧ R)

Tożsamości takie jak te mogą być użyte do zmiany wyrażeń rachunku zdań w odmienną składniowo, ale logicznie równoważną formę. Tych tożsamości można użyć zamiast tabel prawdy, aby udowodnić, że dwa wyrażenia są równoważne: znaleźć serię tożsamości, które przekształcają jedno wyrażenie w drugie. Wczesny program sztucznej inteligencji, Teoretyk Logiki, zaprojektowany przez Newella, Simona i Shawa, wykorzystali transformacje między równoważnymi formami wyrażeń, aby udowodnić wiele twierdzeń Whiteheada i Russella, Principia Mathematica (1950). Zdolność do zmiany logicznego wyrażenia na inną formę o równoważnych wartościach prawdy jest również ważna, gdy stosuje się reguły wnioskowania , które wymagają, aby wyrażenia były w formie specyficznej.

Rachunek predykatów

W rachunku zdań każdy symbol atomowy (P, Q itd.) oznacza pojedyncze zdanie. Nie ma możliwości uzyskania dostępu do składników pojedynczego stwierdzenia. Rachunek predykatów zapewnia tę zdolność. Na przykład, zamiast pozwolić jednemu symbolowi zdaniowemu, P, oznaczać całe zdanie "padało we wtorek", możemy stworzyć predykatową pogodę, która opisuje związek między datą a pogodą: pogoda (wtorek, deszcz). Dzięki regułom wnioskowania możemy manipulować wyrażeniami rachunku predykatów, uzyskując dostęp do ich poszczególnych składników i wprowadzając nowe zdania. Rachunek predykatów pozwala również, aby wyrażenia zawierały zmienne. Zmienne pozwalają nam tworzyć ogólne twierdzenia o klasach bytów. Na przykład możemy stwierdzić, że dla wszystkich wartości X, gdzie X jest dniem tygodnia, wyrażenie pogoda (X, deszcz) jest prawdziwe; czyli pada codziennie. Podobnie jak w przypadku rachunku zdań, najpierw zdefiniujemy składnię języka, a następnie omówimy jego semantykę.

Składnia predykatów i zdań

Przed zdefiniowaniem składni poprawnych wyrażeń w rachunku predykatów definiujemy alfabet i gramatykę do tworzenia symboli języka. Odpowiada to leksykalnemu aspektowi definicji języka programowania. Symbole rachunku predykatu, podobnie jak tokeny w języku programowania, są nieredukowalnymi elementami składniowymi: nie można ich rozbić na części składowe za pomocą operacji języka. W naszej prezentacji reprezentujemy symbole rachunku predykatu jako ciągi liter i cyfr rozpoczynające się na literę. Nie mogą pojawiać się spacje i znaki niealfanumeryczne wewnątrz ciągu znaków, chociaż podkreślenie _ może być użyte do poprawy czytelności.

DEFINICJA

SYMBOLE RACHUNKU PREDYKATÓW

Alfabet, który składa się z symboli rachunku predykatu, składa się z:

1. Zestaw liter alfabetu angielskiego, zarówno wielkich, jak i małych.

2. Zbiór cyfr, 0, 1, …, 9.

3. Podkreślenie, _.

Symbole w rachunku predykatów zaczynają się od litery, po której następuje dowolna sekwencja tych legalnych znaków. Do prawidłowych znaków alfabetu symboli rachunku predykatu należą

a R 6 9 p _ z

Przykłady znaków spoza alfabetu obejmują

#% @ / &

Uzasadnione symbole rachunku predykatu obejmują

George fire3 tom_and_jerry bill XXXX friends_of

Przykłady ciągów znaków, które nie są legalnymi symbolami

3jack bez pustych miejsc ab% cd *** 71 kaczka !!!

Symbole, jak widzimy , są używane do oznaczania obiektów, właściwości lub relacji w świecie dyskursu. Podobnie jak w przypadku większości języków programowania, użycie "słów" sugerujących zamierzone znaczenie symbolu pomaga nam zrozumieć kod programu. Tak więc, chociaż l (g, k) i polubienia (George, Kate) są formalnie równoważne (tj. mają tę samą strukturę), drugi może być bardzo pomocny (dla ludzkich czytelników) we wskazywaniu, jaką relację reprezentuje to wyrażenie. Należy podkreślić, że te opisowe nazwy służą wyłącznie poprawie czytelności wyrażeń. Jedyne "znaczenie", które mają wyrażenia rachunku predykatu, podane jest poprzez ich formalną semantykę. Nawiasy "()", przecinki "," i kropki "." Są używane wyłącznie do konstruowania dobrze sformułowanych wyrażeń i nie oznaczają obiektów ani relacji na świecie. Są to tak zwane niewłaściwe symbole. Symbole rachunku predykatu mogą reprezentować zmienne, stałe, funkcje lub predykaty. Stałe nazywają określone obiekty lub właściwości na świecie. Symbole stałe muszą zaczynać się małą literą. Tak więc George, drzewo, wysoki i niebieski są przykładami dobrze uformowanych stałych symboli. Stałe prawda i fałsz są zastrzeżone jako symbole prawdy. Symbole zmienne są używane do oznaczania ogólnych klas obiektów lub właściwości na świecie. Zmienne są reprezentowane przez symbole rozpoczynające się od dużej litery. A zatem George, BILL i KAte są zmiennymi prawnymi, podczas gdy George i Bill nie. Rachunek predykatów umożliwia także funkcje na obiektach w świecie dyskursu. Symbole funkcji (jak stałe) zaczynają się od małej litery. Funkcje oznaczają odwzorowanie jednego lub więcej elementów w zestawie (zwanym domeną funkcji) na unikalny element drugiego zestawu (zakres funkcji). Elementami dziedziny i zasięgu są obiekty w świecie dyskursu. Oprócz typowych funkcji arytmetycznych, takich jak dodawanie i mnożenie, funkcje mogą definiować odwzorowania między domenami nieliczbowymi. Zauważ, że nasza definicja symboli rachunku predykatu nie obejmuje liczb ani operatorów arytmetycznych. System liczbowy nie jest uwzględniony w operacjach pierwotnych rachunku predykatu; zamiast tego jest definiowany aksjomatycznie przy użyciu "czystego" rachunku predykatów jako podstawy. Chociaż szczegóły tej pochodnej mają znaczenie teoretyczne, są one mniej ważne dla zastosowania rachunku predykatów jako języka reprezentacji AI. Dla wygody przyjmujemy to wyprowadzenie i uwzględniamy arytmetykę w języku. Każdy symbol funkcji ma powiązaną aranżację, wskazując liczbę elementów w domenie odwzorowanych na każdy element zakresu. W ten sposób ojciec mógłby określić funkcję arity 1, która mapuje ludzi na ich (unikalnego) męskiego rodzica. plus może być funkcją arity 2, która odwzorowuje dwie liczby na ich sumę arytmetyczną. Wyrażenie funkcyjne to symbol funkcji, po której następują jej argumenty. Argumenty są elementami z dziedziny funkcji; liczba argumentów jest równa arity funkcji. Argumenty są ujęte w nawiasy i oddzielone przecinkami. Na przykład,

f(X, Y)

ojciec(dawid)

cena(banany)

są dobrze sformułowanymi wyrażeniami funkcyjnymi.

Każde wyrażenie funkcji oznacza odwzorowanie argumentów na pojedynczy obiekt w zakresie, zwany wartością funkcji. Na przykład, jeśli ojciec jest funkcją jednoargumentową, to ojciec (David) jest wyrażeniem funkcji, którego wartością (w świecie dyskursu autora) jest George. Jeśli plus jest funkcją arity 2, z domeną liczb całkowitych, to plus (2,3) jest wyrażeniem funkcyjnym, którego wartością jest liczba całkowita 5. Czynność zastąpienia funkcji jej wartością nazywana jest oceną. Pojęcie symbolu lub terminu rachunku predykatu sformalizowano w następującej definicji:

DEFINICJA

SYMBOLE I WARUNKI

Symbole rachunku predykatu obejmują:

1. Symbole prawdy prawda i fałsz (są to symbole zastrzeżone).

2. Symbole stałe to wyrażenia symboli, których pierwsza litera jest mała.

3. Symbole zmienne są wyrażeniami symboli rozpoczynającymi się od wielkich liter.

4. Symbole funkcyjne to wyrażenia symboliczne zawierające pierwszą literę małej litery. Funkcje mają dołączony arian wskazujący liczbę elementów domeny odwzorowanych na każdy element zakresu.

Wyrażenie funkcyjne składa się ze stałej funkcji arity n, po której następuje n terminów, t_1, t_2 ... t_n, otoczonych nawiasami i oddzielonych przecinkami. Termin rachunku predykatu jest wyrażeniem stałym, zmiennym lub funkcyjnym. Zatem predykcyjny rachunek różniczkowy może być użyty do oznaczenia obiektów i właściwości w dziedzinie problemowej. Przykłady terminów to:

kot

razy(2,3)

X

niebieski

matka(Sarah)

Kate

Symbole w rachunku predykatów mogą również reprezentować predykaty. Symbole predykatów, takie jak stałe i nazwy funkcji, zaczynają się od małej litery. Predykat określa związek między zero lub większą liczbą obiektów na świecie. Tak powiązana liczba obiektów jest rodzajem predykatu. Przykładami predykatów są polubienia równe w pobliżu części_ Zdanie atomowe, najbardziej prymitywna jednostka języka rachunku predykatów, jest predykatem arity n, po którym następuje n terminów zamkniętych w nawiasach i oddzielonych przecinkami. Przykładami zdań atomowych są

likes(george,kate) likes(X,george)

likes(george,susie) likes(X,X)

likes(george,sarah,tuesday) friends(bill,richard)

friends(bill,george) friends(father_of(david),father_of(andrew))

helps(bill,george) helps(richard,bill)

Symbole predykatów w tych wyrażeniach to polubienia, przyjaciele i pomoc. Symbol predykatu może być używany z różną liczbą argumentów. W tym przykładzie są dwa różne polubienia, jeden z dwoma, a drugi z trzema argumentami. Gdy w zdaniach o różnych aracjach używany jest symbol predykatu, uważa się, że reprezentuje on dwie różne relacje. Tak więc relacja predykatu jest definiowana przez jej nazwę i aranżację. Nie ma powodu, dla którego te dwa różne upodobania nie mogą stanowić części tego samego opisu świata; jednak zwykle tego się unika, ponieważ często może powodować zamieszanie. W powyższych predykatach rachunek, George, Kate itp. Są stałymi symbolami i reprezentują obiekty w dziedzinie problemowej. Argumentami predykatu są terminy i mogą również zawierać zmienne lub wyrażenia funkcyjne. Na przykład przyjaciele (father_of (David), father_of (andrew)) jest predykatem opisującym związek między dwoma obiektami w dziedzinie dyskursu. Argumenty te są reprezentowane jako wyrażenia funkcyjne, których odwzorowania (biorąc pod uwagę, że ojciec_david to George, a ojciec_ andrew to allen) tworzą parametry. Jeśli wyrażenia funkcyjne są oceniane, wyrażenie staje się przyjaciółmi (George, Allen). Te pomysły zostały sformalizowane w następującej definicji.

DEFINICJA

PREDYKAT iI ZDARZENIA ATOMOWE

Symbole predykatów to symbole rozpoczynające się od małej litery.

Predykaty mają powiązaną dodatnią liczbę całkowitą określaną jako arity lub "liczba argumentów" dla predykatu. Predykaty o tej samej nazwie, ale różnych ariach są uważane za odrębne. Zdanie atomowe jest predykatem stałej arity n, po której następuje n terminów, ujętych w nawiasy i oddzielonych przecinkami. Wartości prawdy, prawda i fałsz, są również zdaniami atomowymi.

Zdania atomowe nazywane są również wyrażeniami atomowymi, atomami lub zdaniami. Możemy łączyć zdania atomowe za pomocą operatorów logicznych, aby tworzyć zdania w rachunku predykatów. Są to te same logiczne łączniki, które są używane w rachunku zdań:

∧, ∨, ¬, →, i ≡

Kiedy zmienna pojawia się jako argument w zdaniu, odnosi się do nieokreślonych obiektów w domenie. Rachunek predykatu pierwszego rzędu zawiera dwa symbole, kwantyfikatory zmiennych ∀ i ∃, które ograniczają znaczenie zdania zawierającego zmienną. Po kwantyfikatorze następuje zmienna i zdanie, takie jak

∃Y friends(Y, peter)

∀X likes(X, ice_cream)

Uniwersalny kwantyfikator ∀ wskazuje, że zdanie jest prawdziwe dla wszystkich wartości zmiennej. W przykładzie ∀X polubień (X, lody_kremowe) jest prawdziwe dla wszystkich wartości w dziedzinie definicji X. Egzystencjalny kwantyfikator ∃ wskazuje, że zdanie jest prawdziwe dla co najmniej jednej wartości w domenie. Friends Przyjaciele Y (Y, Peter) są prawdziwe, jeśli istnieje co najmniej jeden obiekt wskazany przez Y, który jest przyjacielem Petera. Zdania w rachunku predykatów są definiowane indukcyjnie.

DEFINICJA

PREYKAT RACHUNKU ZDAŃ

Każde zdanie atomowe jest zdaniem.

1. Jeśli s jest zdaniem, to i jego negacja, ¬s.

2. Jeśli s_1 i s_2 są zdaniami, to także ich koniunkcja, $s_1 \wedge s_2$.

3. Jeśli s_1 i s_2 są zdaniami, to i ich rozłączenie, $s_1 \vee s_2$.

4. Jeśli s_1 i s_2 są zdaniami, to ich implikacja, $s_1 \rightarrow s_2$.

5. Jeśli s_1 i s_2 są zdaniami, to ich równoważność, $s_1 \equiv s_2$.

6. Jeśli X jest zmienną a s zdaniem, to ∀ X s jest zdaniem.

7. Jeśli X jest zmienną a s zdaniem, to ∃ X s jest zdaniem.

Poniżej podano przykłady poprawnie sformułowanych zdań. Niech czasy i plus będą symbolami funkcyjnymi arity 2, a równe i foo będą predykatami symboli odpowiednio arity 2 i 3.

plus(dwa, trzy) jest funkcją, a zatem nie jest zdaniem atomowym.

równe(plus(dwa, trzy), pięć) to zdanie atomowe.

równe(plus(2, 3), siedem) to zdanie atomowe. Zauważ, że to zdanie, biorąc pod uwagę standardową interpretację plus i równość, jest fałszywe. Dobra forma i wartość prawdy są niezależnymi zagadnieniami.

∃Xfoo (X, dwa, plus (dwa, trzy)) ∧ równe (plus (dwa, trzy), pięć) to zdanie, ponieważ obie koniunkcje są zdaniami.

(foo(dwa, dwa, plus (dwa, trzy))) → (równe (plus (trzy, dwa), pięć) ≡ prawda) to zdanie, ponieważ wszystkie jego elementy są zdaniami, odpowiednio połączonymi operatorami logicznymi.

Definicja zdań rachunku predykatu i właśnie przedstawione przykłady sugerują metodę weryfikacji, czy wyrażenie jest zdaniem. Jest to zapisywane jako algorytm rekurencyjny, Verse_sentence. Verit_sentence przyjmuje jako argument wyrażenie kandydujące i zwraca sukces, jeśli wyrażenie jest zdaniem.

function verify_sentence(expression);

begin

case

expression is an atomic sentence: return SUCCESS;

expression is of the form Q X s, where Q is either ∀ or ∃, X is a variable,

if verify_sentence(s) returns SUCCESS

then return SUCCESS

else return FAIL;

expression is of the form ¬ s:

if verify_sentence(s) returns SUCCESS

then return SUCCESS

else return FAIL;

expression is of the form s_1 op s_2, where op is a binary logical operator:

if verify_sentence(s_1) returns SUCCESS and

verify_sentence(s_2) returns SUCCESS

then return SUCCESS

else return FAIL;

otherwise: return FAIL

end

end.

Kończymy tę sekcję przykładem użycia rachunku predykatu do opisu prostego świata. Dziedziną dyskursu jest zbiór relacji rodzinnych w biblijnej genealogii:

mother(eve,abel)

mother(eve,cain)

father(adam,abel)

father(adam,cain)

$\forall X \forall Y$ father(X, Y) $\vee$ mother(X, Y) $\rightarrow$ parent(X, Y)

$\forall X \forall Y \forall Z$ parent(X, Y) $\wedge$ parent(X, Z) $\rightarrow$ sibling(Y, Z)

W tym przykładzie używamy predykatów matka i ojciec, aby zdefiniować zestaw relacji rodzic-dziecko. Implikacje podają ogólne definicje innych relacji, takich jak rodzic i rodzeństwo, w odniesieniu do tych predykatów. Intuicyjnie jasne jest, że implikacje te można wykorzystać do wnioskowania o faktach, takich jak rodzeństwo (cain, abel). Aby sformalizować ten proces, aby można go było przeprowadzić na komputerze, należy dołożyć starań, aby zdefiniować algorytmy wnioskowania i upewnić się, że takie algorytmy rzeczywiście wyciągają prawidłowe wnioski z zestawu twierdzeń rachunku predykatu. W tym celu definiujemy semantykę rachunku predykatów, a następnie rozwiązujemy problem reguł wnioskowania.

Semantyka dla rachunku predykatów

Po zdefiniowaniu dobrze sformułowanych wyrażeń w rachunku predykatów ważne jest, aby określić ich znaczenie w kategoriach obiektów, właściwości i relacji na świecie. Semantyka rachunku predykatów zapewnia formalną podstawę do określenia wartości prawdy poprawnie sformułowanych wyrażeń. Prawda wyrażeń zależy od mapowania stałych, zmiennych, predykatów i funkcji na obiekty i relacje w dziedzinie dyskursu. Prawda relacji w dziedzinie określa prawdziwość odpowiednich wyrażeń. Na przykład informacje na temat osoby, George'a i jego przyjaciół Kate i Susie mogą być wyrażone przez

przyjaciele (George, Susie)

przyjaciele (George, Kate)

Jeśli to prawda, że George jest przyjacielem Susie, a George jest przyjacielem Kate, wówczas każde z tych wyrażeń miałoby wartość prawdy (przypisanie) T. Jeśli George jest przyjacielem Susie, ale nie Kate, wówczas pierwsze wyrażenie byłoby mają wartość prawdy T, a druga miałaby wartość prawdy F. Aby wykorzystać rachunek predykatów jako reprezentację do rozwiązywania problemów, opisujemy obiekty i relacje w dziedzinie interpretacji za pomocą zestawu dobrze sformułowanych wyrażeń. Terminy i predykcje tych wyrażeń oznaczają obiekty i relacje w domenie. Ta baza wyrażeń rachunku predykatu, z których każde ma wartość prawdy T, opisuje "stan świata". Opis George'a i jego przyjaciół jest prostym przykładem takiej bazy danych. Innym przykładem jest świat bloków we wstępie do części II. W oparciu o te intuicje formalnie definiujemy semantykę rachunku predykatów. Najpierw definiujemy interpretację w domenie D. Następnie używamy tej interpretacji do określenia przypisania zdań prawdy do wartości w języku.

DEFINICJA

INTERPRETACJA

Niech domena D będzie niepustym zestawem.

Interpretacja nad D jest przypisaniem bytów D do każdego z symboli stałych, zmiennych, predykatów i funkcji wyrażenia rachunku predykatów, tak że:

1. Każda stała ma przypisany element D.

2. Każda zmienna jest przypisana do niepustego podzbioru D; są to dopuszczalne podstawienia dla tej zmiennej.

3. Każda funkcja f arity m jest zdefiniowana na m argumentach D i definiuje odwzorowanie z D^m na D.

4. Każdy predykat ar ar n jest zdefiniowany na n argumentach z D i definiuje odwzorowanie z D^n na {T, F}.

Biorąc pod uwagę interpretację, znaczenie wyrażenia jest przypisaniem wartości prawdy nad interpretacją.

DEFINICJA

WARTOŚĆPRAWDY WYRAŻEŃ RAHUNKU ZDAŃ

Załóżmy wyrażenie E i interpretację I dla E w niepustej domenie D. Wartość prawdy dla E jest określona przez:

1. Wartość stałej jest elementem D, do którego przypisuje ją I.

2. Wartością zmiennej jest zbiór elementów D, do którego przypisuje ją I.

3. Wartością wyrażenia funkcyjnego jest ten element D uzyskany przez ocenę funkcji dla wartości parametrów przypisanych przez interpretację.

4. Wartość symbolu prawdy "prawda" to T, a "fałsz" to F.

5. Wartość zdania atomowego wynosi T lub F, zgodnie z interpretacją I.

6. Wartość negacji zdania wynosi T, jeśli wartością zdania jest F, a F, jeśli wartością zdania jest T.

7. Wartość koniunkcji dwóch zdań wynosi T, jeśli wartość obu zdań to T, a w przeciwnym razie F.

8. - 10. Wartość prawdziwości wyrażeń używających ∨, →, i ≡ określa się na podstawie wartości ich argumentów określonych w sekcji wcześniejszej. Wreszcie dla zmiennej X i zdania S zawierającego X:

11. Wartość ∀ X S wynosi T, jeśli S jest T dla wszystkich przypisań do X w ramach I, a F w przeciwnym razie.

12. Wartość ∃ X S wynosi T, jeżeli w interpretacji istnieje przyporządkowanie do X; w przeciwnym razie jest to F.

Kwantyfikacja zmiennych jest ważną częścią semantyki rachunku predykatu. Kiedy zmienna pojawia się w zdaniu, na przykład X w polubieniach (George, X), zmienna działa jako symbol zastępczy. Każda stała dozwolona w ramach interpretacji może zostać zastąpiona w wyrażeniu. Podstawienie kate lub susie za X w polubieniach (George, X) tworzy stwierdzenia polubienia (George, Kate) i polubienia (George, Susie), jak widzieliśmy wcześniej. Zmienna X oznacza wszystkie stałe, które mogą pojawić się jako drugi parametr zdania. Ta nazwa zmiennej może zostać zastąpiona dowolną inną nazwą zmiennej, taką jak Y lub LUDZIE, bez zmiany znaczenia wyrażenia. Zatem zmienna jest uważana za manekina. W rachunku predykatów zmienne muszą być kwantyfikowane na jeden z dwóch sposobów: uniwersalnie lub egzystencjalnie. Zmienna jest uważana za swobodną, jeżeli nie wchodzi w zakres kwantyfikatorów uniwersalnych lub egzystencjalnych. Wyrażenie jest zamykane, jeśli wszystkie jego zmienne są kwantyfikowane. Wyrażenie podstawowe nie ma żadnych zmiennych. W rachunku predykatów wszystkie zmienne muszą być kwantyfikowane. Nawiasy są często używane w celu wskazania zakresu kwantyfikacji, to znaczy wystąpień nazwy zmiennej, które dotyczą kwantyfikacji. Zatem dla symbolu wskazującego uniwersalną kwantyfikację ∀:

$$\forall X\,(p(X) \vee q(Y) \rightarrow r(X))$$

wskazuje, że X jest powszechnie określany ilościowo zarówno w p(X), jak i r(X). Uniwersalna kwantyfikacja wprowadza problemy w obliczaniu prawdziwej wartości zdania, ponieważ wszystkie możliwe wartości symbolu zmiennej muszą zostać przetestowane, aby sprawdzić, czy wyrażenie pozostaje prawdziwe. Na przykład, aby przetestować wartość prawdy likes X polubień (George, X), gdzie X rozciąga się na zbiór wszystkich ludzi, należy przetestować wszystkie możliwe wartości X. Jeśli dziedzina interpretacji jest nieskończona, wyczerpujące testowanie wszystkich podstawień zmiennej uniwersalnej jest niemożliwe obliczeniowo: algorytm nigdy nie może zostać zatrzymany. Z tego powodu mówi się, że rachunek predykatów jest nierozstrzygalny. Ponieważ rachunek zdań nie obsługuje zmiennych, zdania mogą mieć tylko skończoną liczbę przypisań prawdy i możemy wyczerpująco przetestować wszystkie te możliwe przypisania. Dokonuje się tego za pomocą tabeli prawdy. Zmienne mogą być również egzystencjalnie określone ilościowo, oznaczone symbolem ?. W przypadku egzystencjalnym mówi się, że wyrażenie zawierające zmienną jest prawdziwe dla co najmniej jednego podstawienia z jego dziedziny definicji. Zakres egzystencjalnie skwantyfikowanej zmiennej jest również wskazany poprzez umieszczenie skwantyfikowanych wystąpień zmiennej w nawiasach. Ocena prawdziwości wyrażenia zawierającego zmienną skwantyfikowaną egzystencjalnie może nie być łatwiejsze niż ocena prawdziwości wyrażeń zawierających zmienne skwantyfikowane. Załóżmy, że próbujemy ustalić prawdziwość wyrażenia, próbując podstawienia, dopóki nie zostanie znalezione takie, które sprawi, że wyrażenie będzie prawdziwe. Jeśli domena zmiennej jest nieskończona a wyrażenie jest fałszywe przy wszystkich podstawieniach, algorytm nigdy się nie zatrzyma. Kilka zależności między negacją a uniwersalnymi i egzystencjalnymi kwantyfikatorami podano poniżej. Zależności te są wykorzystywane w opisanych powyżej systemach odrzucania rozdzielczości. Zwraca się również uwagę na nazwę zmiennej jako symbol zastępczy, który oznacza zbiór stałych. Dla predykatów p i q oraz zmiennych X i Y:

$$\neg \exists X\, p(X) \equiv \forall X \neg p(X)$$

¬ ∀ X p(X) ≡ ∃ X ¬ p(X)

∃ X p(X) ≡ ∃ Y p(Y)

∀ X q(X) ≡ ∀ Y q(Y)

∀ X (p(X) ∧ q(X)) ≡ ∀ X p(X) ∧ ∀ Y q(Y)

∃ X (p(X) ∨ q(X)) ≡ ∃ X p(X) ∨ ∃ Y q(Y)

W zdefiniowanym przez nas języku uniwersalnie i egzystencjalnie zmienne kwantyfikowane mogą odnosić się tylko do obiektów (stałych) w dziedzinie dyskursu. Predykatów i nazw funkcji nie można zastępować zmiennymi kwantyfikowanymi. Ten język nosi nazwę rachunek predykatów pierwszego rzędu

DEFINICJA

RACHUNEK PREDYKATÓW PIERWSZEGO RZĘDU

Rachunek predykatów pierwszego rzędu pozwala zmiennym kwantyfikowanym odwoływać się do obiektów w dziedzinie dyskursu, a nie do predykatów lub funkcji.

Na przykład ∀ (Likes) Likes (George, Kate) nie jest dobrze sformułowanym wyrażeniem w rachunku predykatów pierwszego rzędu. Istnieją wyliczenia predykatów wyższego rzędu, w których takie wyrażenia są znaczące. Niektórzy badacze używali języków wyższego rzędu do reprezentowania wiedzy w programach do rozumienia języka naturalnego. Wiele poprawnych gramatycznie zdań w języku angielskim może być reprezentowanych w rachunku predykatów pierwszego rzędu za pomocą symboli, łączników i symboli zmiennych zdefiniowanych w tej sekcji. Należy zauważyć, że nie ma unikalnego mapowania zdań na wyrażenia rachunku predykatu; w rzeczywistości zdanie angielskie może mieć dowolną liczbę różnych reprezentacji rachunku predykatu. Głównym wyzwaniem dla programistów AI jest znalezienie schematu wykorzystania tych predykatów, który optymalizuje ekspresję i wydajność wynikowej reprezentacji. Przykłady zdań w języku angielskim reprezentowanych w rachunku predykatów są:

Jeśli w poniedziałek nie pada, Tom pojedzie w góry.

¬ pogoda (deszcz, poniedziałek) → idź (tom, góry)

Emma to Doberman Pinczer i dobry pies.

gooddog (emma) ∧ isa (emma, doberman)

Wszyscy koszykarze są wysocy.

∀ X (gracz koszykówki (X) → wysoki (X))

Niektórzy ludzie lubią sardele.

∃ X (osoba (X) ∧ lubi (X, anchois))

Gdyby życzenia były koniami, żebracy jeździliby

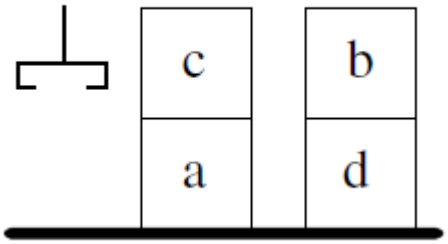
równe (życzenia, konie) → jazda (żebracy)

Nikt nie lubi podatków.

¬ ∃ Polubienia (X, podatki)

Przykład "świata bloków" o znaczeniu semantycznym

Kończymy tę sekcję, podając rozszerzony przykład przypisania wartości prawdy do zestawu wyrażeń rachunku predykatu. Załóżmy, że chcemy modelować świat bloków z rysunku w celu zaprojektowania, na przykład, algorytmu sterowania dla ramienia robota.



Możemy użyć zdań rachunku predykatu do przedstawienia relacji jakościowych na świecie: czy dany blok ma wyraźną górną powierzchnię? czy możemy odebrać blok a? itp. Załóżmy, że komputer ma wiedzę o położeniu każdego bloku i ramienia i jest w stanie śledzić te położenia (przy użyciu trójwymiarowych współrzędnych), gdy ręka przesuwa bloki wokół stołu. Musimy być bardzo precyzyjni w kwestii tego, co proponujemy na tym przykładzie "świata bloków". Po pierwsze, tworzymy zestaw wyrażeń rachunku predykatu, który ma reprezentować statyczną migawkę domeny problemu świata bloków. Jak zobaczymy później, ten zestaw bloków oferuje interpretację i możliwy model zbioru wyrażeń rachunku predykatu. Po drugie, rachunek predykatów jest deklaratywny, to znaczy nie ma założonego terminu ani kolejności uwzględnienia każdego wyrażenia. Niemniej jednak w sekcji planowania , dodamy "semantykę proceduralną" lub jasno określoną metodologię oceny tych wyrażeń w czasie. Konkretnym przykładem semantyki proceduralnej dla wyrażeń rachunku predykatu jest Prolog. Ten rachunek sytuacji, który tworzymy, wprowadzi szereg zagadnień, w tym problem ramowy i zagadnienie niemonotoniczności interpretacji logicznych, które zostaną omówione w dalszej części tej książki. W tym przykładzie wystarczy jednak powiedzieć, że nasze wyrażenia rachunku predykatu będą oceniane od góry do dołu i od lewej do prawej. Aby podnieść blok i ułożyć go w innym bloku, oba bloki muszą być czyste. Na rysunku , blok a nie jest jasny. Ponieważ ramię może przesuwać bloki, może zmieniać stan świata i usuwać blok. Załóżmy, że usuwa blok c z bloku a i aktualizuje bazę wiedzy, aby to odzwierciedlić, usuwając twierdzenie z (c, a). Program musi być w stanie wywnioskować, że blok a stał się jasny. Poniższa reguła opisuje, kiedy blok jest wyczyszczony:

$\forall$ X ($\neg$ $\exists$ Y on (Y, X) $\rightarrow$ wyczyść (X))

Oznacza to, że dla wszystkich X, X jest jasne, jeśli nie istnieje Y takie, że Y jest na X. Ta reguła nie tylko określa, co to znaczy, że blok jest czysty, ale także stanowi podstawę do ustalenia, jak usunąć bloki, które nie są. Na przykład blok d nie jest jasny, ponieważ jeśli zmienna X otrzyma wartość d, podstawienie b na Y spowoduje, że instrukcja będzie fałszywa. Dlatego, aby ta definicja była prawdziwa, blok b należy usunąć z bloku d. Można to łatwo zrobić, ponieważ komputer ma zapis wszystkich bloków i ich lokalizacji. Oprócz użycia implikacji do zdefiniowania, kiedy blok jest czysty, można dodać inne reguły opisujące operacje, takie jak układanie jednego bloku na drugim. Na przykład: aby ułożyć X na Y, najpierw opróżnij rękę, następnie wyczyść X, następnie wyczyść Y, a następnie pick_up X i umieść X na Y.

$\forall$ X $\forall$ Y ((hand_empty $\wedge$ clear(X) $\wedge$ clear(Y) $\wedge$ pick_up(X) $\wedge$ put_down(X,Y)) $\rightarrow$stack(X,Y))

Zauważ, że przy implementacji powyższego opisu konieczne jest "dołączenie" działania ramienia robota do każdego predykatu, takiego jak pick_up (X). Jak wspomniano wcześniej, dla takiej implementacji konieczne było zwiększenie semantyki rachunku predykatów poprzez wymaganie, aby działania były wykonywane w kolejności, w jakiej występują w przesłance reguły. Jednak wiele zyskuje się poprzez oddzielenie tych zagadnień od użycia rachunku predykatu do zdefiniowania relacji i operacji w domenie. Rysunek powyższy przedstawia interpretację semantyczną tych wyrażeń rachunku predykatu. Ta interpretacja odwzorowuje stałe i predykaty w zestawie wyrażeń na domenę D, tutaj bloki i relacje między nimi. Interpretacja nadaje wartości T wartość każdemu wyrażeniu w opisie. Inną interpretację może zaoferować inny zestaw klocków w innym miejscu lub zespół czterech akrobatów. Ważnym pytaniem nie jest wyjątkowość interpretacji, ale to, czy interpretacja zapewnia wartość prawdy dla wszystkich wyrażeń w zbiorze i czy wyrażenia opisują świat wystarczająco szczegółowo, aby można było przeprowadzić wszystkie niezbędne wnioskowania poprzez manipulację wyrażeniami symbolicznymi. W następnej sekcji wykorzystano te pomysły, aby zapewnić formalną podstawę reguł wnioskowania rachunku predykatu.

Używanie reguł wnioskowania do tworzenia wyrażeń rachunku predykatu

Zasady wnioskowania

Semantyka rachunku predykatów stanowi podstawę formalnej teorii wnioskowania logicznego. Zdolność wnioskowania nowych poprawnych wyrażeń na podstawie zestawu prawdziwych twierdzeń jest ważną cechą rachunku predykatów. Te nowe wyrażenia są w tym poprawne są one zgodne ze wszystkimi poprzednimi interpretacjami oryginalnego zestawu wyrażeń. Najpierw omawiamy te pomysły nieoficjalnie, a następnie tworzymy zestaw definicji, aby były precyzyjne. Mówi się, że interpretacja, która sprawia, że zdanie jest zgodne z prawdą. Mówi się, że interpretacja, która spełnia każdy element zestawu wyrażeń, spełnia zestaw. Wyrażenie X logicznie wynika z zestawu wyrażeń rachunku predykatów S, jeśli każda interpretacja, która spełnia S, również spełnia X. To pojęcie daje nam podstawę do weryfikacji poprawności reguł wnioskowania: funkcją wnioskowania logicznego jest tworzenie nowych zdań, które logicznie postępuj zgodnie z danym zestawem zdań rachunku różniczkowego. Ważne jest, aby zrozumieć dokładne znaczenie logicznie następującego: aby wyrażenie X logicznie następowało po S, musi być prawdą w przypadku każdej interpretacji, która spełnia oryginalny zestaw wyrażeń S. Oznaczałoby to na przykład, że każde nowe wyrażenie rachunku predykatu dodane do świata bloków z rysunku 2.3 musi być prawdziwe w tym świecie, a także w każdej innej interpretacji, jaką może mieć ten zestaw wyrażeń. Sam termin "logicznie podąża" może być nieco mylący. Nie oznacza to, że X można wywnioskować z, ani nawet, że można go wydedukować ze S. To po prostu oznacza, że X jest prawdą dla każdej interpretacji spełniającej S. Jednak, ponieważ systemy predykatów mogą mieć potencjalnie nieskończoną liczbę możliwych interpretacji, jest to rzadko praktyczne jest wypróbowanie wszystkich interpretacji. Zamiast tego reguły wnioskowania zapewniają obliczeniowo wykonalny sposób ustalenia, kiedy wyrażenie, składnik interpretacji, logicznie podąża za tą interpretacją. Pojęcie "logicznie podąża" stanowi formalną podstawę dla dowodów prawidłowości i poprawności reguł wnioskowania. Reguła wnioskowania jest w istocie mechanicznym środkiem do tworzenia nowych zdań predykatowych z innych zdań. Oznacza to, że reguły wnioskowania tworzą nowe zdania na podstawie składniowej formy podanych twierdzeń logicznych. Gdy każde zdanie X wytworzone przez regułę wnioskowania działającą na zbiorze S wyrażeń logicznych logicznie wynika z S, o reguły wnioskowania mówi się, że jest solidna. Jeśli reguła wnioskowania jest w stanie wygenerować każde wyrażenie logicznie wynikające z S, to mówi się, że jest kompletne. Modus ponens, który zostanie wprowadzony poniżej, oraz rozdzielczość, wprowadzona w rozdziale 11, są przykładami zasad wnioskowania, które są rozsądne i, jeśli są stosowane z odpowiednimi strategiami aplikacji, kompletne. Logiczne systemy wnioskowania na ogół używają rozsądnych reguł wnioskowania, chociaż

późniejsze części badają rozumowanie heurystyczne i rozumowanie zdrowego rozsądku, które rozluźniają ten wymóg. Formalizujemy te pomysły za pomocą następujących definicji.

DEFINICJA

ZADOWOLONY, MODELOWY, WAŻNY, NIEZGODNY

W przypadku rachunku predykatu wyrażenie X i interpretacja I:

Jeśli X ma wartość T w ramach I i szczególne przypisanie zmiennej, to mówi się, że spełniam X.

Jeśli spełniam X dla wszystkich przypisań zmiennych, to jestem modelem X.

X jest zadowalające tylko wtedy, gdy istnieje interpretacja i przypisanie zmiennych, które go spełniają; w przeciwnym razie jest niezadowalający.

Zestaw wyrażeń jest zadowalający tylko wtedy, gdy istnieje interpretacja i przypisanie zmiennych, które spełniają każdy element.

Jeśli zestaw wyrażeń nie jest zadowalający, mówi się, że jest niespójny.

Jeśli X ma wartość T dla wszystkich możliwych interpretacji, mówi się, że X jest poprawny.

W przykładzie świata bloków z rysunku powyżej świat bloków był modelem jego logicznego opisu. Wszystkie zdania w tym przykładzie były zgodne z tą interpretacją. Gdy baza wiedzy jest implementowana jako zbiór prawdziwych stwierdzeń na temat problematycznej domeny, domena ta jest modelem dla bazy wiedzy. Wyrażenie $\exists X (p(X) \wedge \neg p(X))$ jest niespójne, ponieważ nie może być spełnione przy żadnej interpretacji lub przypisaniu zmiennej. Z drugiej strony prawidłowe jest wyrażenie $\forall X (p(X) \vee \neg p(X))$.

Metodę tabeli prawdy można wykorzystać do przetestowania poprawności dowolnego wyrażenia niezawierającego zmiennych. Ponieważ nie zawsze jest możliwe ustalenie ważności wyrażeń zawierających zmienne (jak wspomniano powyżej, proces może się nie zakończyć), pełny rachunek predykatów jest "nierozstrzygalny". Istnieją jednak procedury dowodowe, które mogą wygenerować dowolne wyrażenie, które logicznie następuje z zestawu wyrażeń. Są to tak zwane procedury pełnego dowodu.

DEFINICJA

PROCEDURA DOWODOWA

Procedura sprawdzająca jest kombinacją reguły wnioskowania i algorytmu do zastosowania tej reguły do zestawu wyrażeń logicznych w celu wygenerowania nowych zdań. Później przedstawiamy procedury sprawdzające regułę wnioskowania o rozstrzygnięciu. Korzystając z tych definicji, możemy formalnie zdefiniować "logicznie podąża".

DEFINICJA

LOGICZNE PODĄŻANIE, BRZMIENIE I KOMPLET

Wyrażenie rachunku predykatów X logicznie wynika ze zbioru S wyrażeń rachunku predykatów, jeżeli każda interpretacja i przypisanie zmiennych, które spełnia S, również spełnia X. Reguła wnioskowania jest skuteczna, jeśli każde wyrażenie rachunku predykatu wygenerowane przez regułę ze zbioru S wyrażeń rachunku predykatu również logicznie wynika ze S. Reguła wnioskowania jest kompletna, jeśli biorąc pod uwagę zbiór S wyrażeń rachunku predykatu, reguła może wywnioskować każde wyrażenie, które logicznie wynika z S. Modus ponens jest regułą wnioskowania dźwiękowego. Jeśli otrzymamy

wyrażenie formy P → Q i inne wyrażenie formy P, takie, że oba są prawdziwe zgodnie z interpretacją I, wówczas modus ponens pozwala nam wnioskować, że Q jest również prawdziwe dla tej interpretacji. Rzeczywiście, ponieważ modus ponens jest dźwiękiem, Q jest prawdziwe dla wszystkich interpretacji, dla których P i P → Q są prawdziwe. Modus ponens i szereg innych przydatnych reguł wnioskowania określono poniżej.

DEFINICJA

MODUS PONENS, MODUS TOLLENS I ELIMINACJA ORAZ WPROWADZENIE I UNIWERSALNA INSTALACJA

Jeśli wiadomo, że zdania P i P →Q są prawdziwe, to modus ponens pozwala wywnioskować Q. Zgodnie z regułą wnioskowania modus tollens, jeśli P → Q jest znane jako prawda, a Q wiadomo, że jest fałszem, możemy wywnioskować, że P jest fałszem: ¬P. A eliminacja pozwala nam wywnioskować prawdę jednego z koniunkcji z prawdy zdania łączącego. Na przykład P ∧ Q pozwala stwierdzić, że P i Q są prawdziwe. A wprowadzenie pozwala nam wywnioskować prawdę koniunkcji z prawdy jej koniunkcji. Na przykład, jeśli P i Q są prawdziwe, to P ∧ Q jest prawdziwe. Instancja uniwersalna stwierdza, że jeśli dowolna uniwersalnie skwantyfikowana zmienna w prawdziwym zdaniu, powiedzmy p(X), zostanie zastąpiona odpowiednim terminem z dziedziny, wynikiem jest prawdziwe zdanie. Zatem, jeśli a pochodzi z domeny X, ∀ X p(X) pozwala wywnioskować p(a). Jako prosty przykład zastosowania modus ponens w rachunku zdań, załóżmy następujące spostrzeżenia: "jeśli pada deszcz, wówczas ziemia jest mokra" i "pada deszcz". Jeśli P oznacza "pada deszcz", a Q oznacza " ziemia jest mokra ", wówczas pierwsze wyrażenie zmienia się w P → Q. Ponieważ rzeczywiście teraz pada (P to prawda), nasz zestaw aksjomatów staje się P → QP Dzięki zastosowaniu modus ponens fakt, że ziemia jest mokra (Q) można dodać do zestawu prawdziwych wyrażeń. Modus ponens można również stosować do wyrażeń zawierających zmienne. Rozważmy jako przykład powszechny sylogizm: "wszyscy ludzie są śmiertelni, a Sokrates jest mężczyzną; dlatego Sokrates jest śmiertelny. "" Wszyscy ludzie są śmiertelni "mogą być reprezentowani w rachunku predykatu przez: ∀X (człowiek (X) → śmiertelnik (X)).

"Sokrates jest mężczyzną" to człowiek (Sokrates). Ponieważ X w implikacji jest powszechnie kwantyfikowane, możemy podstawić dowolną wartość w domenę dla X i nadal mają prawdziwe stwierdzenie zgodnie z regułą wnioskowania o uniwersalnej instancji. Zastępując X w implikacji Sokratesa, wnioskujemy o wyrażeniu człowiek (Sokrates) → śmiertelny (Sokrates).

Możemy teraz zastosować modus ponens i wnioskować, że śmiertelnik (Sokrates). Jest to dodawane do zestawu wyrażeń, które logicznie wynikają z pierwotnych asercji. Algorytm zwany unifikacją może zostać wykorzystany przez zautomatyzowane narzędzie do rozwiązywania problemów w celu ustalenia, że Sokrates może zastąpić X w celu zastosowania modus ponens.

Unifikacja

Aby zastosować reguły wnioskowania, takie jak modus ponens, system wnioskowania musi być w stanie określić, kiedy dwa wyrażenia są takie same lub pasują. W rachunku zdań jest to trywialne: dwa wyrażenia pasują tylko wtedy, gdy są identyczne pod względem składniowym. W rachunku predykatów proces dopasowywania dwóch zdań jest skomplikowany przez istnienie zmiennych w wyrażeniach. Uniwersalne tworzenie instancji pozwala na zastąpienie uniwersalnych zmiennych ilościowych terminami z dziedziny. Wymaga to procesu decyzyjnego w celu ustalenia podstawień zmiennych, w ramach których dwa lub więcej wyrażeń może być identycznych (zwykle w celu zastosowania reguł wnioskowania).

Ujednolicenie jest algorytmem służącym do określania podstawień potrzebnych do dopasowania dwóch wyrażeń rachunku predykatu. Widzieliśmy to już w poprzednim podrozdziale, w którym

Sokrates u człowieka (Sokrates) zastąpiono X w ∀ X(człowiek(X) ⇒ śmiertelny(X)). To pozwoliło na zastosowanie modus ponens i wniosek śmiertelnika (Sokrates). Inny przykład unifikacji zaobserwowano wcześniej, gdy omawiano zmienne fikcyjne. Ponieważ p(X) i p(Y) są równoważne, Y może zastąpić X, aby dopasować zdania. Reguły unifikacji i wnioskowania, takie jak modus ponens, pozwalają nam wnioskować na podstawie zestawu logicznych twierdzeń. Aby to zrobić, logiczna baza danych musi być wyrażona w odpowiedniej formie. Zasadniczym aspektem tej formy jest wymaganie, aby wszystkie zmienne były uniwersalne i skwantyfikowane. Pozwala to na pełną swobodę w zastępowaniu obliczeń. Zmienne ilościowe można wyeliminować ze zdań w bazie danych, zastępując je stałymi, dzięki którym zdanie jest prawdziwe. Na przykład parent X rodzic (X, tom) może zostać zastąpiony przez wyrażenie rodzic (Bob, Tom) lub rodzic (Mary, Tom), zakładając, że Bob i Mary są rodzicami Toma w interpretacji. Proces eliminowania egzystencjalnie skwantyfikowanych zmiennych jest skomplikowany przez fakt, że wartość tych podstawień może zależeć od wartości innych zmiennych w wyrażeniu. Na przykład w wyrażeniu ∀ X ∃ Matka Y (X, Y) wartość egzystencjalnie skwantyfikowanej zmiennej Y zależy od wartości X. Skolemizacja zastępuje każdą egzystencjalnie skwantyfikowaną zmienną funkcją, która zwraca odpowiednią stałą w funkcji niektóre lub wszystkie inne zmienne w zdaniu. W powyższym przykładzie, ponieważ wartość Y zależy od X, Y można zastąpić funkcją skolem, f, z X. Daje to predykat mother X matkę (X, f (X)). Skolemizacja, proces, który może również wiązać zmienne o wartościach uniwersalnych ze stałymi. Po usunięciu istniejących ilościowo zmiennych z logicznej bazy danych można zastosować unifikację do dopasowania zdań w celu zastosowania reguł wnioskowania, takich jak modus ponens. Ujednolicenie komplikuje fakt, że zmienną można zastąpić dowolnym terminem, w tym innymi zmiennymi i wyrażeniami funkcji o dowolnej złożoności. Te wyrażenia mogą same zawierać zmienne. Na przykład ojciec (walet) może zastąpić X u mężczyzny (X), aby wywnioskować, że ojciec waletera jest śmiertelny. Niektóre wystąpienia wyrażenia foo (X, a, goo (Y)). generowane przez prawne zastępstwa podano poniżej:
1) foo (fred, a, goo (Z))

2) foo (W, a, goo (jack))

3) foo (Z, a, goo (moo (Z)))

W tym przykładzie wystąpienia lub ujednolicenia podstawienia, które spowodowałyby, że początkowe wyrażenie byłoby identyczne z każdym z pozostałych trzech, zapisano jako zestawy:

1) {fred / X, Z / Y}

2) {W / X, jack / Y}

3) {Z / X, moo (Z) / Y}

Notacja X / Y,... wskazuje, że X zastępuje zmienną Y w pierwotnym wyrażeniu. Podstawienia są również nazywane wiązaniami. Mówi się, że zmienna jest związana z podstawioną wartością. Przy definiowaniu algorytmu unifikacji, który oblicza podstawienia wymagane do dopasowania dwóch wyrażeń, należy wziąć pod uwagę szereg kwestii. Po pierwsze, chociaż stała może być systematycznie zastępowana zmienną, każda stała jest uważana za "instancję podstawową" i nie może być zastąpiona. Ani jednej zmiennej nie można zastąpić dwoma różnymi instancjami naziemnymi. Po drugie, zmiennej nie można ujednolicić z terminem zawierającym tę zmienną. X nie może być zastąpione przez p (X), ponieważ tworzy to nieskończone wyrażenie: p (p (p (p (p (… X) …))). Test na tę sytuację nazywa się sprawdzaniem zachodzącym. Ponadto proces rozwiązywania problemów często wymaga wielu wnioskowania, a w konsekwencji wielu kolejnych unifikacji. Rozwiązania problemów logicznych muszą zachować spójność podstawień zmiennych. Ważne jest, aby wszelkie podstawienia ujednolicające były konsekwentnie wykonywane we wszystkich wystąpieniach w zakresie zmiennej w dopasowywanych obu wyrażeniach. Było to widoczne wcześniej, gdy Sokrates zastąpiono nie tylko zmienną X u człowieka (X), ale także zmienną X u śmiertelnika (X). Po związaniu zmiennej przyszłe ujednolicenia i wnioski muszą

uwzględniać wartość tego wiązania. Jeśli zmienna jest powiązana ze stałą, zmienna ta może nie otrzymać nowego wiązania w przyszłej unifikacji. Jeśli zmienna X1 zostanie zastąpiona inną zmienną X2, a później X1 zostanie zastąpiona stałą, wówczas X2 musi również odzwierciedlać to wiązanie. Zestaw podstawień zastosowany w sekwencji wnioskowania jest ważny, ponieważ może zawierać odpowiedź na pierwotne zapytanie. Na przykład, jeśli p (a, X) łączy się z założeniem p (Y, Z) ⇒ q (Y, Z) z podstawieniem {a / Y, X / Z}, modus ponens pozwala nam wnioskować q (a, X) w ramach tego samego zastąpienia. Jeśli dopasujemy ten wynik do założenia q (W, b) ⇒ r (W, b), wnioskujemy r (a, b) w ramach zestawu podstawień {a / W, b / X}. Inną ważną koncepcją jest skład podstawień unifikacyjnych. Jeśli S i S 'są dwoma zestawami podstawień, wówczas skład S i S' (zapisane SS ') uzyskuje się przez zastosowanie S' do elementów S i dodanie wyniku do S. Rozważ przykład komponowania następujących trzech zestawów podstawień:

{X / Y, W / Z}, {V / X}, {a / V, f (b) / W}.

Komponowanie trzeciego zestawu {a / V, f (b) / W} z drugim {V / X} daje:

{a / X, a / V, f (b) / W}.

Skomponowanie tego wyniku z pierwszym zestawem {X / Y, W / Z} daje zestaw podstawień:

{a / Y, a / X, a / V, f (b) / Z, f (b) / W}.

Kompozycja to metoda łączenia podstawień unifikacyjnych i zwracania ich w funkcji rekurencyjnej unify, przedstawiona poniżej. Kompozycja jest asocjacyjna, ale nie przemienna. Ćwiczenia przedstawiają te kwestie bardziej szczegółowo. Kolejnym wymogiem algorytmu unifikacji jest to, aby unifikator był jak najbardziej ogólny: aby znaleźć najbardziej ogólny unifikator. Jest to ważne, jak zobaczymy w następnym przykładzie, ponieważ jeśli ogólność zostanie utracona w procesie rozwiązania, może zmniejszyć zakres ostatecznego rozwiązania lub nawet całkowicie wyeliminować możliwość rozwiązania. Na przykład w ujednolicaniu p(X) i p(Y) będzie działać dowolne wyrażenie stałe, takie jak {fred / X, fred / Y}. Jednak fred nie jest najbardziej ogólnym unifikatorem; każda zmienna dałaby bardziej ogólne wyrażenie: {Z / X, Z / Y}. Rozwiązania uzyskane z pierwszej instancji podstawienia byłyby zawsze ograniczone przez stałe ograniczenie prędkości wynikowych wniosków; tj. fred byłby unifikatorem, ale zmniejszyłby ogólność wyniku

DEFINICJA

NAJBARDZIEJ OGÓLNY UNIFIKATOR (mgu)

Jeśli s jest dowolnym unifikatorem wyrażeń E, a g jest najbardziej ogólnym unifikatorem tego zestawu wyrażeń, to dla s zastosowanego do E istnieje inny unifikator s ? taki, że Es = Egs', gdzie Es i Egs' są kompozycją unifikatory zastosowane do wyrażenia E. Najbardziej ogólny unifikator dla zestawu wyrażeń jest unikalny, z wyjątkiem odmian alfabetycznych; tj. to, czy zmienna zostanie ostatecznie nazwana X, czy Y, tak naprawdę nie ma żadnego wpływu na ogólność wynikowych unifikacji. Ujednolicenie jest ważne dla każdego rozwiązania problemu sztucznej inteligencji, który używa rachunku predykatu do reprezentacji. Ujednolicenie określa warunki, w których można powiedzieć, że dwa (lub więcej) wyrażenia rachunku predykatu są równoważne. Pozwala to na stosowanie reguł wnioskowania, takich jak rozwiązywanie, z reprezentacjami logicznymi, co często wymaga cofnięcia w celu znalezienia wszystkich możliwych interpretacji. Następnie przedstawiamy pseudo-kod dla funkcji unify, która może obliczać podstawienia unifikujące (jeśli jest to możliwe) między dwoma wyrażeniami rachunku predykatu. Unify przyjmuje jako argumenty dwa wyrażenia w rachunku predykatów i zwraca albo najbardziej ogólny zestaw podstawień unifikujących, albo stały FAIL, jeśli żadna unifikacja nie jest możliwa. Jest zdefiniowany jako funkcja rekurencyjna: po pierwsze, rekurencyjnie próbuje ujednolicić

początkowe komponenty wyrażeń. Jeśli to się powiedzie, wszelkie podstawienia zwrócone przez tę unifikację zostaną zastosowane do reszty obu wyrażeń. Są one następnie przekazywane w drugim rekurencyjnym wezwaniu do unifikacji, które próbuje zakończyć unifikację. Rekurencja kończy się, gdy dowolny argument jest symbolem (predykatem, nazwą funkcji, stałą lub zmienną) lub gdy wszystkie elementy wyrażenia zostały dopasowane. Aby uprościć manipulację wyrażeniami, algorytm zakłada nieco zmodyfikowaną składnię. Ponieważ unify po prostu wykonuje składniowe dopasowanie wzorca, może skutecznie ignorować rozróżnienie rachunku predykatów między predykatami, funkcjami i argumentami. Reprezentując wyrażenie jako listę (uporządkowaną sekwencję elementów) z nazwą predykatu lub funkcji jako pierwszym elementem, a następnie jego argumentami, upraszczamy manipulację wyrażeniami. Wyrażenia, w których sam argument jest predykatem lub wyrażeniem funkcyjnym, są reprezentowane jako listy na liście, zachowując w ten sposób strukturę wyrażenia. Listy są rozdzielane nawiasami (), a elementy listy są oddzielane spacjami. Przykłady wyrażeń zarówno w rachunku predykatów, na PC, jak i na liście:

PC SYNTAX	LIST SYNTAX
p(a,b)	(p a b)
p(f(a),g(X,Y))	(p (f a) (g X Y))
equal(eve,mother(cain))	(equal eve (mother cain))

We next present the function unify:

```
function unify(E1, E2);
  begin
    case
      both E1 and E2 are constants or the empty list:          %recursion stops
        if E1 = E2 then return {}
          else return FAIL;
      E1 is a variable:
        if E1 occurs in E2 then return FAIL
          else return (E2/E1);
      E2 is a variable:
        if E2 occurs in E1 then return FAIL
          else return (E1/E2)
      either E1 or E2 are empty then return FAIL     %the lists are of different sizes
      otherwise:                                          %both E1 and E2 are lists
        begin
            HE1 := first element of E1;
            HE2 := first element of E2;
            SUBS1 := unify(HE1,HE2);
            if SUBS1 : = FAIL then return FAIL;
            TE1 := apply(SUBS1, rest of E1);
            TE2 : = apply (SUBS1, rest of E2);
            SUBS2 : = unify(TE1, TE2);
            if SUBS2 = FAIL then return FAIL;
                else return composition(SUBS1,SUBS2)
        end
    end                                                          %end case
  end
```

Przykład unifikacji

Zachowanie powyższego algorytmu można wyjaśnić, śledząc wywołanie unify ((rodzice X (ojciec X) (rachunek matki)), (rachunek rodziców (rachunek ojca) Y)). Kiedy unify jest wywoływane po raz pierwszy, ponieważ żaden argument nie jest symbolem atomowym, funkcja spróbuje rekurencyjnie ujednolicić pierwsze elementy każdego wyrażenia, wywołując

unify(rodzice, rodzice).

To zjednoczenie się powiodło, zwracając puste podstawienie {}. Zastosowanie tego do pozostałych wyrażeń nie powoduje żadnych zmian; algorytm następnie wywołuje

unify ((X (ojciec X) (rachunek matki)), (rachunek (rachunek ojca) Y)).

W drugim wywołaniu unifikacji żadne wyrażenie nie jest atomowe, więc algorytm dzieli każde wyrażenie na pierwszy składnik i pozostałą część wyrażenia. To prowadzi do połączenia

unify (X, rachunek).

Wywołanie to się powiedzie, ponieważ oba wyrażenia są atomowe, a jedno z nich jest zmienną. Wywołanie zwraca podstawienie {bill / X}. Podstawienie to stosuje się do reszty każdego wyrażenia, a wyniki unify są wywoływane w wynikach:

unify (((rachunek ojca) (rachunek matki)), ((rachunek ojca) Y)).

Wynikiem tego połączenia jest ujednolicenie (rachunek ojca) z (rachunek ojca). To prowadzi do połączeń

unify (ojciec, ojciec)

unify (rachunek, rachunek)

unify ((), ())

Wszystko to się udaje, zwracając pusty zestaw podstawień. Następnie wywoływana jest funkcja Unify dla pozostałych wyrażeń:

unify (((rachunek główny)), (Y)).

To z kolei prowadzi do połączeń

unify ((rachunek główny), Y)

unify ((), ()).

W pierwszym z nich (rachunek macierzysty) łączy się z Y. Zwróć uwagę, że unifikacja zastępuje całą strukturę (rachunek macierzysty) zmienną Y. Tak więc unifikacja kończy się powodzeniem i zwraca podstawienie {(rachunek główny) / Y}. Wywołanie

unify ((), ())

zwraca { }. Wszystkie podstawienia są tworzone po zakończeniu każdego połączenia rekurencyjnego, aby zwrócić odpowiedź {bill / X (mother bill) / Y}. Każde połączenie jest ponumerowane, aby wskazać kolejność, w jakiej zostało wykonane; podstawienia zwracane przez każde wywołanie są odnotowywane na łukach drzewa.

Aplikacja: Doradca finansowy oparty na logice

Jako ostatni przykład wykorzystania rachunku predykatów do reprezentowania i uzasadniania domen problemowych, projektujemy doradcę finansowego przy użyciu rachunku predykatu. Chociaż jest to prosty przykład, ilustruje on wiele problemów związanych z realistycznymi aplikacjami. Zadaniem doradcy jest pomoc użytkownikowi w podjęciu decyzji, czy zainwestować w rachunek oszczędnościowy czy giełdę. Niektórzy inwestorzy mogą chcieć podzielić swoje pieniądze między nimi. Inwestycja, która będzie zalecana inwestorom indywidualnym, zależy od ich dochodów i bieżącej kwoty, którą zaoszczędzili, zgodnie z następującymi kryteriami:

1. Osoby z nieodpowiednim kontem oszczędnościowym powinny zawsze zwiększać zaoszczędzoną kwotę na pierwszym miejscu, niezależnie od swoich dochodów.

2. Osoby posiadające odpowiedni rachunek oszczędnościowy i odpowiedni dochód powinny rozważyć bardziej ryzykowną, ale potencjalnie bardziej opłacalną inwestycję na giełdzie.

3. Osoby o niższych dochodach, które mają już odpowiedni rachunek oszczędnościowy, mogą rozważyć podzielenie nadwyżki dochodu na oszczędności i zapasy, aby zwiększyć poduszkę oszczędności, próbując jednocześnie zwiększyć swoje dochody poprzez zapasy.

Adekwatność zarówno oszczędności, jak i dochodu zależy od liczby osób na utrzymaniu, które jednostka musi wspierać. Naszą zasadą jest co najmniej 5000 USD oszczędności na każdego członka rodziny. Odpowiedni dochód musi być stałym dochodem i zapewniać co najmniej 15 000 USD rocznie plus dodatkowe 4000 USD na każdego członka rodziny. Aby zautomatyzować tę poradę, tłumaczymy te wytyczne na zdania w rachunku predykatu. Pierwszym zadaniem jest określenie głównych cech, które należy wziąć pod uwagę. Tutaj są adekwatnością oszczędności i dochodu. Są one reprezentowane odpowiednio przez rachunek oszczędnościowy i dochód. Oba są jednoznacznymi predykatami, a ich argument może być odpowiedni lub nieodpowiedni. A zatem,

savings_account(adequate).

savings_account(inadequate).

income(adequate).

income(inadequate).

są ich możliwymi wartościami.

Wnioski są reprezentowane przez jednostkową inwestycję predykatową, przy czym możliwymi wartościami tego argumentu są zapasy, oszczędności lub połączenie (co oznacza, że inwestycja powinna zostać podzielona). Stosując te predykaty, różne strategie inwestycyjne są reprezentowane przez implikacje. Pierwsza zasada, że osoby z niewystarczającymi oszczędnościami powinny poczynić do wzrostu oszczędności jest ich głównym priorytetem, jest reprezentowany przez

savings_account(inadequate) → investment(savings)

Podobnie pozostałe dwie możliwe alternatywy inwestycyjne są reprezentowane przez

savings_account(adequate) ∧ income(adequate) → investment(stocks).

savings_account(adequate) ∧ income(inadequate) → investment(combination).

Następnie doradca musi ustalić, kiedy oszczędności i dochód są odpowiednie lub niewystarczające. Zostanie to również wykonane przy użyciu implikacji. Konieczność wykonywania obliczeń arytmetycznych wymaga użycia funkcji. Aby określić minimalne odpowiednie oszczędności, zdefiniowano funkcję oszczędzania min. minsavings pobiera jeden argument, liczbę osób zależnych i zwraca 5000 razy ten argument. Stosując oszczędności minimalne, adekwatność oszczędności zależy od zasad

∀ X amount_saved(X) ∧ ∃ Y (dependents(Y) ∧ greater(X, minsavings(Y)))

→ savings_account(adequate).

∀ X amount_saved(X) ∧ ∃ Y (dependents(Y) ∧ ¬ greater(X, minsavings(Y)))

→ savings_account(inadequate).

gdzie minsavings (X) ≡ 5000 ? X.

W tych definicjach amount_saved (X) i osoby na dependents(Y) potwierdzają bieżącą kwotę oszczędności i liczbę osób na utrzymaniu inwestora; greater(X, Y) jest standardowym testem

arytmetycznym dla jednej liczby większej od drugiej i nie jest formalnie zdefiniowana w tym przykładzie. Podobnie, funkcja minincome jest zdefiniowana jako

minincome (X) ≡ 15000 + (4000 ? X).

minincome służy do obliczenia minimalnego odpowiedniego dochodu, biorąc pod uwagę liczbę osób na utrzymaniu. Bieżące dochody inwestora reprezentowane są przez predykat, zyski. Ponieważ odpowiedni dochód musi być zarówno stały, jak i powyżej minimum, zarobki wymagają dwóch argumentów: pierwszy to kwota zarobiona, a drugi musi być równy albo stały, albo niestały. Pozostałe zasady potrzebne doradcy to:

∀ X earnings(X, steady) ∧ ∃ Y (dependents(Y) ∧ greater(X, minincome(Y))) → income(adequate).

∀ X earnings(X, steady) ∧ ∃ Y (dependents(Y) ∧ ¬ greater(X, minincome(Y))) → income(inadequate).

∀ X earnings(X, unsteady) → income(inadequate).

W celu przeprowadzenia konsultacji do tego zestawu zdań rachunku predykatu dodawany jest opis konkretnego inwestora z wykorzystaniem predykatów kwota_zapisana, zarobki i osoby na utrzymaniu. Tak więc osoba z trzema osobami na utrzymaniu, o oszczędności 22 000 USD i stałym dochodzie w wysokości 25 000 USD, opisałaby

amount_saved(22000)

earnings(25000, steady).

dependents(3)

To daje logiczny system składający się z następujących zdań:

1. savings_account(inadequate) → investment(savings).

2. savings_account(adequate) ∧ income(adequate) → investment(stocks).

3. savings_account(adequate) ∧ income(inadequate)

→ investment(combination).

4. ∀ X amount_saved(X) ∧ ∃ Y (dependents(Y) ∧

greater(X, minsavings(Y))) → savings_account(adequate).

5. ∀ X amount_saved(X) ∧ ∃ Y (dependents(Y) ∧

¬ greater(X, minsavings(Y))) → savings_account(inadequate).

6. ∀ X earnings(X, steady) ∧ ∃ Y (dependents (Y) ∧

greater(X, minincome(Y))) → income(adequate).

7. ∀ X earnings(X, steady) ∧ ∃ Y (dependents(Y) ∧

¬ greater(X, minincome(Y))) → income(inadequate).

8. ∀ X earnings(X, unsteady) → income(inadequate).

9. amount_saved(22000).

10. earnings(25000, steady).

11. dependents(3).

gdzie minsavings (X) ≡ 5000 * X i minincome (X) ≡ 15000 + (4000 * X). Ten zestaw zdań logicznych opisuje domenę problemu. Asercje są ponumerowane, dzięki czemu można się do nich odwoływać w następującym śladzie. Używając unifikacji i modus ponens, właściwą strategię inwestycyjną dla tej osoby można wywnioskować jako logiczną konsekwencję tych opisów. Pierwszym krokiem byłoby ujednolicenie połączenia 10 i 11 z pierwszymi dwoma składnikami założenia 7; to znaczy.,

earnings(25000,steady) ∧ dependents(3)

unifies with

earnings(X,steady) ∧ dependents(Y)

pod podstawieniem {25000 / X, 3 / Y}. Ta zamiana daje nową implikację:

earnings(25000, steady) ∧ dependents(3) ∧ ¬ greater(25000, minincome(3))

→ income(inadequate).

Ocena funkcji minincome daje wyrażenie

earnings(25000, steady) ∧ dependents(3) ∧ ¬ greater(25000, 27000)

→ income(inadequate).

Ponieważ wszystkie trzy składniki przesłanki są indywidualnie prawdziwe, o 10, 3, a matematyczna definicja większej, ich połączenie jest prawdziwe, a cała przesłanka jest prawdziwa. Można zatem zastosować modus ponens, co da dochód z wniosku (nieodpowiedni). Jest to dodane jako twierdzenie 12.

12. income(inadequate).

Podobnie,

amount_saved(22000) ∧ dependents(3)

łączy się z pierwszymi dwoma elementami założenia asercji 4 pod podstawieniem {22000 / X, 3 / Y}, dając implikację

amount_saved(22000) ∧ dependents(3) ∧ greater(22000, minsavings(3))

→ savings_account(adequate).

Tutaj ocena funkcji minsavings (3) daje wyrażenie

amount_saved(22000) ∧ dependents(3) ∧ greater(22000, 15000)

→ savings_account(adequate)

Ponownie, ponieważ wszystkie składniki przesłanki tej implikacji są prawdziwe, cała przesłanka ocenia się na prawdę, a modus ponens można ponownie zastosować, uzyskując wniosek o oszczędności_konta (odpowiedni), który dodaje się jako wyrażenie 13.

13. savings_account(adequate).

Jak wskazuje analiza wyrażeń 3, 12 i 13, przesłanka implikacji 3 jest również prawdziwa. Kiedy ponownie zastosujemy modus ponens, wnioskiem jest inwestycja (połączenie), sugerowana inwestycja

dla naszej osoby. Ten przykład ilustruje, w jaki sposób można zastosować rachunek predykatu do uzasadnienia realistycznego problemu, wyciągając prawidłowe wnioski, stosując reguły wnioskowania do wstępnego opisu problemu. Nie dyskutowaliśmy dokładnie, w jaki sposób algorytm może określić prawidłowe wnioski, aby rozwiązać dany problem, ani sposób, w jaki można to zaimplementować na komputerze.

Epilog

Wprowadziliśmy rachunek predykatów jako język reprezentacyjny do rozwiązywania problemów AI. Symbole, terminy, wyrażenia i semantyka języka zostały opisane i zdefiniowane. W oparciu o semantykę rachunku predykatów zdefiniowaliśmy reguły wnioskowania, które pozwalają nam wyprowadzać zdania, które logicznie wynikają z danego zestawu wyrażeń. Zdefiniowaliśmy algorytm unifikacji, który określa podstawienia zmiennych, które dopasowują dwa wyrażenia, co jest niezbędne do zastosowania reguł wnioskowania. Zakończyliśmy ten rozdział przykładem doradcy finansowego, który reprezentuje wiedzę finansową za pomocą rachunku predykatu i wykazuje logiczne wnioskowanie jako technikę rozwiązywania problemów.