Reprezentacja wiedzy w języku logiki

Metody przeszukiwania w przestrzeni stanów sformułowane były w postaci dość ogólnej, jednak wymagały reprezentacji zagadnienia we właściwej formie, tzn. przestrzeni stanów, zbioru operatorów, a dodatkowo przydatna/potrzebna była informacja heurystyczna w formie funkcji oceny stanów.

Ogólnie, format i sposób reprezentacji wiedzy o zagadnieniu są niezwykle istotne i mają bezpośredni wpływ na efektywność — lub w ogóle zdolność — znalezienia rozwiązania.

Istnieje szereg opracowanych ogólnych podejść do problemu reprezentacji, i różne reprezentacje mają zwykle związane z nimi techniki **wnioskowania**, czyli formowania pewnych ustaleń pomocniczych (wniosków), mogących służyć do znalezienia ostatecznego rozwiązania problemu.

Jednym z najpopularniejszych schematów reprezentacji wiedzy jest język logiki matematycznej.

Dlaczego logika matematyczna jest dobrym językiem reprezentacji wiedzy dla sztucznej inteligencji?

Z jednej strony, język logiki jest zbliżony do sposobu w jaki ludzie myślą o świecie, i myśli swe wyrażają w zdaniach języka naturalnego. Czasami mówi się kolokwialnie, że człowiek myśli „logicznie". Kategorie, którymi myśli i mówi człowiek obejmują takie konstrukcje jak: obiekty, związki między obiektami (relacje), stwierdzenia faktów prostych i złożonych, zdania, spójniki zdaniowe, wyrażenia faktów warunkowych, a nawet kwantyfikatory.

Z drugiej strony, logika matematyczna dostarcza precyzyjnego aparatu wnioskowania opartego na dowodzeniu twierdzeń. Ludzie, myśląc, również stosują podobne wnioskowanie logiczne, zatem aparat logiki matematycznej wydaje się dobrą platformą reprezentacji wiedzy agenta inteligentnego, którego sposób wyrażania faktów i wnioskowania byłby zbliżony do ludzkiego.

Przykład: świat wumpusa

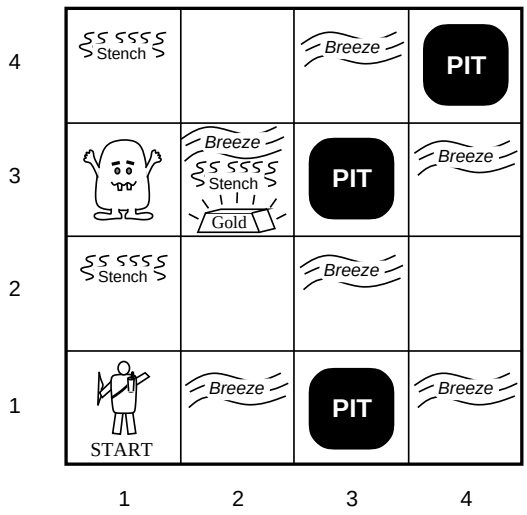
Do sprawdzenia działania wielu metod przydatne jest środowisko testowe dostatecznie proste, aby można było intuicyjnie określać właściwe reprezentacje i sprawdzać proste koncepcje, ale jednocześnie dostatecznie bogate, aby pozwoliło konfrontować te metody z coraz bardziej realnymi przeszkodami.

Jednym z takich testowych środowisk podręcznikowych jest **świat wumpusa**.[1] W tym środowisku porusza się agent dążący do znalezienia złota (i bezpiecznego wyniesienia go z jaskini). Na przeszkodzie stoją zapadliny (*pits*), w które agent może wpaść, i potwór (tytułowy *wumpus*), który może agenta zjeść.

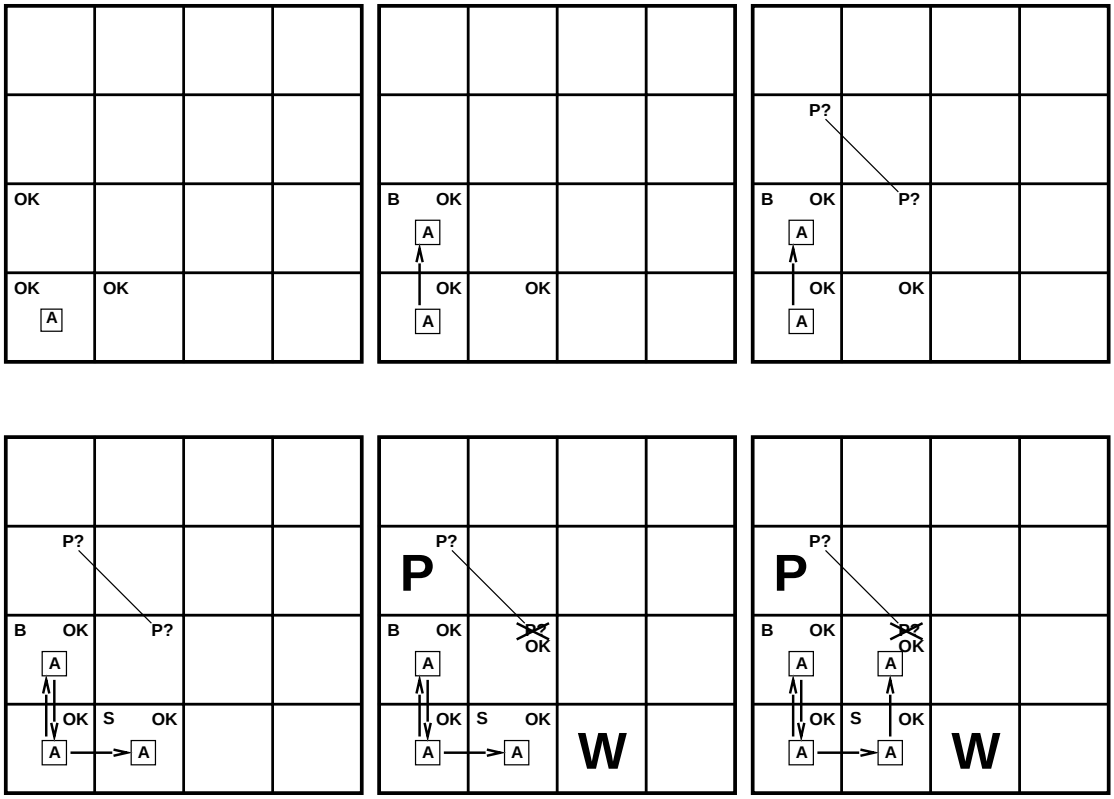
Agent może jedynie obracać się w prawo lub w lewo, poruszać się po jednym kroku do przodu, wystrzelić z łuku jedyną posiadaną strzałę (na wprost), podnieść złoto, gdy je znajdzie, i wyjść z jaskini, jeśli znajduje się w punkcie startowym.

[1] Przedstawione tu przykłady i diagramy świata wumpusa zaczerpnięte zostały z podręcznika Russella i Norviga „Artificial Intelligence A Modern Approach" i materiałów udostępnionych na stronie internetowej Stuarta Russella.

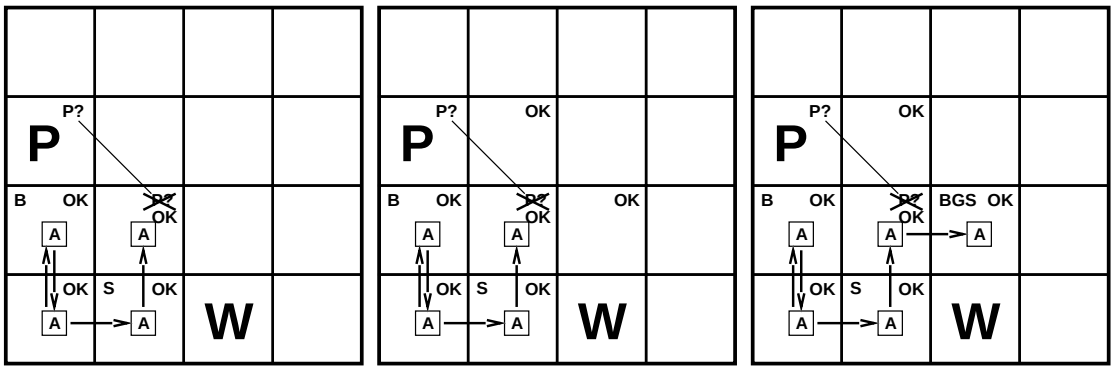
Agent otrzymuje pewne informacje o środowisku (dane otrzymywane z receptorów agenta nazywa się **perceptami**): wyczuwa smród wumpusa (*stench*) i powiew powietrza z zapadlin (*breeze*), jeśli znajduje się w polu sąsiadującym z nimi. Ponadto zauważa złoto (*gold*), ale tylko jeśli jest w tym samym polu co ono. Nie może jednak sprawdzić swojej bezwzględnej pozycji (*à la* GPS), może jedynie sam swoją pozycję rejestrować. Ściany jaskini wyczuwa jedynie przez próby wejścia w nie, które powodują odbicia.



Przykład: poruszanie się w świecie wumpusa



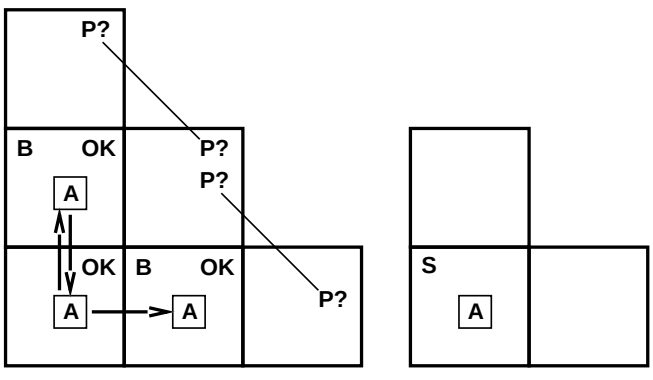
Przykład: poruszanie się w świecie wumpusa (cd.)



Jednak nie zawsze można skutecznie działać w świecie wumpusa posługując się tylko wnioskowaniem logicznym.

W niektórych przypadkach jedynym rozwiązaniem jest „strzelanie", czyli wybór ruchu na ślepo, i dopiero potem analizowanie sytuacji.

Oczywiście o ile przeżyjemy!!



Rachunek predykatów pierwszego rzędu — termy

Termy reprezentują w języku logiki obiekty, i mogą być: stałymi (oznaczają konkretny obiekt), zmiennymi (mogą przybierać wartości różnych obiektów), lub funkcjami (wyznaczają jakiś obiekt na podstawie wartości swoich argumentów, inaczej, przypisują jednym obiektom inne).

Przykłady termów: A, 123, x, $f(A)$, $f(g(x))$, $+(x, 1)$

Umownie zapisujemy termy stałe wielkimi literami, a zmienne małymi.

Zauważmy, że w powyższych przykładach ostatni term jest niejawnym zapisem następnika x, a nie operacją odejmowania. W czystej logice nie ma odejmowania. Będziemy mieli do czynienia z tym problemem w wielu sytuacjach.

Rachunek predykatów pierwszego rzędu — predykaty

Predykaty reprezentują relacje na zbiorze termów. Możemy je traktować jako funkcje mające wartość prawdy lub fałszu (1 lub 0), które przypisują prawdę każdej n-ce termów spełniających relację, a fałsz każdej n-ce niespełniającej.

Zapis predykatu z zestawem termów nazywamy **formułą atomową**.

Przykłady formuł atomowych: P, $Q(A)$, $R(x, f(A))$, $>(x, 10)$

Zapis funkcyjny: $>(x, 10)$ jest odpowiednikiem relacji: $x > 10$. W arytmetyce potraktowalibyśmy taki zapis jako nierówność i moglibyśmy ją rozwiązywać. Natomiast formuły logiczne możemy jedynie **wartościować**, to znaczy wyznaczać ich wartość logiczną prawdy lub fałszu. Gdy formuła zawiera zmienną to często nie da się wyznaczyć jej wartości logicznej.

Reprezentacja faktów za pomocą formuł

Jaki sens ma język logiki predykatów?

Możemy przy jego użyciu opisać fakty, które chcemy wyrazić, np.:

$$At(Wumpus, 2, 2)$$
$$At(Agent, 1, 1)$$
$$At(Gold, 3, 2)$$

Wybór zestawu symboli, którymi zamierzamy opisać obiekty i relacje pewnego świata, nazywamy **konceptualizacją**. Na przykład, alternatywna do powyższej konceptualizacja świata wumpusa mogłaby zawierać formuły:

$$AtWumpus(loc(2, 2))$$
$$AtAgent(loc(1, 1))$$
$$AtGold(loc(3, 2))$$

Powyższe konceptualizacje są podobne, aczkolwiek mają nieco odmienne właściwości, np. w drugiej wumpus, agent, ani złoto nie wystąpią w jawnej postaci. Ogólnie od przyjętej konceptualizacji może zależeć łatwość, a nawet możliwość wyrażania różnych faktów o dziedzinie problemowej.

Reprezentacja faktów za pomocą formuł (cd.)

Przykładem problemu konceptualizacyjnego świata wumpusa jest opis istnienia i położenia dziur. Możemy nadać dziurom prawa obywatelskie i tożsamość:

$$At(Pit_4, 3, 3)$$

W ten sposób możemy łatwo opisać cały świat widziany z lotu ptaka, nadając poszczególnym dziurom dowolnie wybrane nazwy (termy stałe). Z punktu widzenia agenta poruszającego się w świecie wumpusa ta konceptualizacja jest jednak bardzo niewygodna. Trudno byłoby w ten sposób opisać świat stopniowo poznawany, gdy na początku agent nie zna nawet liczby dziur. Istnienie dziury trzeba wtedy opisać zmienną:

$$At(x, 3, 3)$$

Jednak z tego zapisu nie wynika, że x jest dziurą i wymaga to uzupełniających opisów. Wygodną alternatywą jest postrzeganie dziur jako anonimowych, i tylko zapisywanie faktów istnienia lub nieistnienia dziur w konkretnych miejscach:

$$PitAt(3, 3)$$
$$NoPitAt(1, 1)$$

Spójniki logiczne i formuły złożone

Formuły złożone języka predykatów pierwszego rzędu można konstruować za pomocą **spójników logicznych** takich jak: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$. Jako szczególny przypadek, formułę która jest formułą atomową lub negacją formuły atomowej nazywamy **literałem**.
Przykłady formuł złożonych (pierwsza jest pojedynczym literałem):

$$\neg At(Wumpus, 1, 1)$$
$$PitAt(2, 1) \vee PitAt(1, 2)$$
$$[At(Agent, 1, 1) \wedge PitAt(2, 1)] \Rightarrow Percept(Breeze)$$

Zauważmy, że ostatnia formuła jest zupełnie innej natury, niż dwie pierwsze. Dwie pierwsze formuły mogą stanowić fragment opisu świata otrzymanego i/lub budowanego przez agenta inteligentnego w trakcie jego pracy w świecie wumpusa. Natomiast ostatnia formuła wyraża jedno z praw świata wumpusa. Agent zna to prawo, i może posiadać taką formułę w swojej bazie wiedzy.

Fakty ogólnie słuszne w danej dziedzinie problemowej nazywamy **aksjomatami świata**. Natomiast fakty opisujące stan konkretnej instancji problemu, nazywamy **incydentalnymi**.

Warto jeszcze podkreślić, że $\Rightarrow$ i $\Leftrightarrow$ są tylko spójnikami logicznymi, tworzącymi z pary dowolnych formuł formułę złożoną. Nie mają one nic wspólnego z procesem wnioskowania, który będzie rozważany poniżej.

Kwantyfikatory

Formuły złożone można również budować za pomocą **kwantyfikatorów**: $\forall, \exists$, które **wiążą** zmienne w formułach. Ogólny schemat formuły z kwantyfikatorem:

$$\forall x P(x)$$

Zmienną niezwiązaną kwantyfikatorem w formule nazywamy **wolną**. Formuła:

$$\exists y Q(x, y)$$

zawiera dwie zmienne, jedną wolną, a drugą związaną kwantyfikatorem.

Zdaniem nazywamy formułę bez wolnych zmiennych.

Przykłady:

$\exists x, y\ At(Gold, x, y)$
$\forall x, y\ [At(Wumpus, x, y) \wedge At(Agent, x, y)] \Rightarrow AgentDead$
$\forall x, y\ [At(Wumpus, x, y) \wedge At(Agent, -(x, 1), y)] \Rightarrow Percept(Stench)$

Zwróćmy uwagę, że w ostatnim przykładzie $-(x, 1)$ jest niejawnym zapisem współrzędnej na lewo od x, a nie odejmowaniem. W logice nie ma odejmowania.

Krótkie podsumowanie — pytania sprawdzające

1. Opracuj kompletną konceptualizację świata wumpusa w rachunku predykatów pierwszego rzędu. To znaczy: wprowadź symbole termów (stałych i funkcji termowych), oraz symbole predykatów niezbędne do opisywania instancji problemów w tej dziedzinie.

   Uwaga: nie rozważamy procesu poszukiwania rozwiązania, analizy alternatywnych ruchów i ich skutków, opisywania sekwencji kroków, itp. Poszukujemy jedynie formatu pozwalającego na statyczny opis stanu zagadnienia, tzw. *snapshot*.

2. Wykorzystując reprezentację opracowaną w poprzednim punkcie, opisz instancję problemu przedstawioną na stronie 4.

3. Spróbuj zapisać aksjomatykę świata wumpusa, to znaczy: ogólne reguły rządzące tym światem.

Semantyka — interpretacje

Definicja języka predykatów określiła jedną składową reprezentacji, tzn. jego składnię, inaczej **syntaktykę**. Drugą składową jest **semantyka**, czyli aparat pozwalający określać znaczenia.

Na pozór, znaczenie niektórych formuł mogłoby wydawać się jasne. Dotychczas intuicyjnie domyślaliśmy się co znaczy $At(Agent, 2, 2)$. W ogólności to jednak nie wystarczy. Nawet ograniczając się do świata wumpusa, nie wiadomo której tury gry i którego stanu tej gry dotyczy podana formuła.

Interpretacja – przypisanie syntaktycznym elementom języka predykatów (termom i predykatom) obiektów z jakiejś konkretnej dziedziny (świata).

W oczywisty sposób, konkretna formuła zapisana przy użyciu pewnego zestawu symboli może dotyczyć różnych dziedzin. Na przykład, formuła $At(Agent, 2, 2)$ może odnosić się do pewnej tury wumpusa, sceny z filmu o agencie 007, jakiegoś agenta ze świata rzeczywistego, albo jeszcze innego agenta. Możliwych interpretacji dla danej formuły może być bardzo wiele.

Modele

Zauważmy, że interpretacja określa prawdziwość formuł atomowych. Jeśli w danej dziedzinie zachodzi relacja między pewnymi obiektami, to formuła zapisująca to za pomocą odpowiednich termów i predykatu jest prawdziwa w tej interpretacji.

Również na odwrót, mając zapisaną dowolną formułę, możemy wyznaczyć jej wartość logiczną sprawdzając czy w danej interpretacji zachodzi opisany przez formułę związek. (Jednak gdy formuła zawiera zmienne wolne to jej prawdziwość może nie być określona przez interpretację.)

Konsekwentnie, korzystając z definicji spójników logicznych i kwantyfikatorów możemy wyznaczyć wartość logiczną dowolnej formuły (zdania) dla danej interpretacji. Możemy więc mówić, że interpretacje określają prawdziwość formuł zdaniowych (bez zmiennych wolnych).

Interpretację przypisującą danej formule wartość prawdy logicznej nazywamy interpretacją spełniającą, albo **modelem** tej formuły.

Spełnialność

Formułę nazywamy **spełnialną**, jeśli istnieje interpretacja spełniająca, czyli przypisująca jej wartość prawdy logicznej (inaczej: jeśli istnieje model tej formuły). Formuła jest **niespełnialna**, jeśli nie istnieje żadna spełniająca ją interpretacja (model). Z kolei, jeśli formuła ma wartość prawdy dla każdej możliwej interpretacji, to nazywamy ją **tautologią**, albo formułą **prawdziwą**.

Rozważmy przykłady:

$$At(Wumpus, 2, 2)$$
$$\exists x, y\ At(Gold, x, y)$$
$$\forall x, y\ [At(Wumpus, x, y) \wedge At(Agent, x, y)] \Rightarrow AgentDead$$

Wszystkie podane formuły są spełnialne, ponieważ bez trudu możemy stworzyć sobie instancję świata wumpusa, w której będą zachodziły podane fakty. Jednak żadna nie jest tautologią. Co prawda ostatnie dwie formuły są prawdziwe w każdej instancji rozgrywki zgodnej z prawami świata wumpusa. Jednak bez trudu możemy wymyślić inny, podobny świat, gdzie nie będą one obowiązywały.

Przykładem tautologii może być znacznie mniej interesująca, chociaż z matematycznego punktu widzenia fascynująca formuła: $\mathrm{P} \vee \neg \mathrm{P}$, gdzie P jest dowolnym 0-argumentowym predykatem, czyli stwierdzeniem dowolnego faktu.

Krótkie podsumowanie — pytania sprawdzające

1. Dla podanych poniżej formuł rachunku predykatów odpowiedz, czy dana formuła jest: spełnialna, niespełnialna, tautologią.

   (a) P
   (b) $P(x)$
   (c) $\forall x\ P(x)$
   (d) $\exists x\ P(x)$
   (e) $[P \Rightarrow Q] \wedge P \wedge \neg Q$
   (f) $[P \Rightarrow Q] \Leftrightarrow [\neg P \vee Q]$

Wynikanie logiczne

W wielu sytuacjach agent chciałby przeprowadzić jakieś wnioskowanie. Na przykład: w celu stwierdzenia zachodzenia faktów, o których agent nie ma informacji, ale które mogą wynikać z innych informacji, które agent posiada. Takie przypadki widzieliśmy na przykładach ze świata wumpusa. Inną sytuacją gdy agent mógłby chcieć wyciągać wnioski jest próba określenia możliwych skutków swoich działań — pożądanych bądź niepożądanych.

Chcemy mieć możliwość efektywnego sprawdzania, czy jedne fakty wynikają z innych. Jednak logika pozwala jedynie określać wynikanie formuł.

Wynikanie logiczne formuły φ ze zbioru formuł Δ zachodzi, gdy każda interpretacja spełniająca wszystkie formuły z Δ spełnia też formułę φ, co zapisujemy:

$$\Delta \models \varphi$$

Dlaczego właśnie tak definiujemy wynikanie? Bo chcemy mieć uniwersalny aparat logiczny, prowadzący procesy wnioskowania poprawne dla wszystkich możliwych interpretacji. Ponieważ operujemy na formułach, chcemy mieć pewność, że proces wnioskowania jest słuszny również dla konkretnej dziedziny problemowej, z która ma do czynienia agent.

Dygresja — rachunek zdań

W logice istnieje język prostszy od rachunku predykatów zwany **rachunkiem zdań**. Nie istnieją w nim termy, zatem predykaty ograniczone są do predykatów 0-argumentowych, zwanych zdaniami. Formuły atomowe stanowią pojedyncze symbole zdaniowe, a formuły złożone można budować za pomocą spójników, podobnie jak w rachunku predykatów. Kwantyfikatorów nie ma.

Semantyka rachunku zdań jest bardzo uproszczona. Zamiast rozpatrywać wszystkie możliwe interpretacje danej formuły wystarczy podzielić je na grupy, w których poszczególne symbole zdaniowe są prawdziwe lub nieprawdziwe. W efekcie, zamiast rozpatrywać interpretacje, można badać wszystkie możliwe kombinacje prawdy i fałszu symboli zdaniowych. Prostą procedurą takiego badania jest budowanie tabelek zerojedynkowych.

W rachunku predykatów nie jest to możliwe, ponieważ wartość logiczna formuły zależy od wartości termów, które są argumentami predykatów. Niestety, rachunek zdań jest zbyt słaby (mówimy: niedostatecznie **ekspresyjny**) dla zastosowań praktycznych.

Reguły wnioskowania

Sprawdzanie wynikania przez interpretacje i niespełnialność może być uciążliwe. Jest to spowodowane koniecznością uwzględnienia wszystkich możliwych interpretacji, których może być bardzo dużo.

Zamiast sprawdzania wynikania logicznego, w logice stosuje się podejście zwane **dowodzeniem twierdzeń**. Wprowadza się **reguły wnioskowania**, które pozwalają z jednych formuł tworzyć inne za pomocą syntaktycznych przekształceń.

Na przykład, jedna z podstawowych reguł wnioskowania zwana *modus ponens*, albo regułą odrywania, ma postać:

$$\frac{\varphi, \varphi \Rightarrow \psi}{\psi}$$

W przypadku gdyby agent posiadał fakty: $Susza$ i $Susza \Rightarrow NiskiePlony$, to podstawiając je do powyższego schematu miałby prawo wywieść nowy fakt: $NiskiePlony$.

Dowodzenie twierdzeń

Dowodem formuły φ dla zbioru formuł Δ, zwanego zbiorem **aksjomatów**, nazywamy ciąg formuł, z których ostatnią formułą jest φ, a z których każda spełnia jeden z warunków:

1. jest tautologią,
2. jest jednym z aksjomatów,
3. jest formułą otrzymaną z formuł poprzedzających ją w dowodzie (leżących bardziej na lewo) za pomocą jednej z posiadanych reguł wnioskowania.

Twierdzeniem zbioru formuł Δ nazywamy każdą formułę φ która posiada dowód dla zbioru Δ. Wtedy mówimy, że formułę φ można **wywieść** ze zbioru aksjomatów Δ, co zapisujemy:

$$\Delta \vdash \varphi$$

Zbiór wszystkich twierdzeń zbioru formuł Δ nazywamy **teorią** tego zbioru i oznaczamy $\mathcal{T}(\Delta)$.

Wnioskowanie przez dowodzenie twierdzeń

Początkowo wprowadziliśmy wynikanie logiczne ($\Delta \models \varphi$) jako metodę wnioskowania, tzn. stwierdzania wynikania faktów. Jednak brak było wygodnej procedury efektywnego obliczania tego wynikania.

Dowodzenie twierdzeń ($\Delta \vdash \varphi$) oferuje potencjalnie dobrą procedurę wnioskowania dla agenta posługującego się logiką jako językiem reprezentacji. Chcąc wywieść jakiś pożądany fakt ze zbioru aksjomatów można w najgorszym przypadku systematycznie generować wszystkie możliwe skończone ciągi formuł spełniających definicję dowodu tego faktu. Jeśli istnieje dowód o długości N, to w tej procedurze zostanie on wygenerowany. To już lepiej niż sprawdzanie wszystkich interpretacji, które nawet trudno sobie wyobrazić.

Jednak czy dowodzenie twierdzeń jest równie dobrą metodą sprawdzania wynikania faktów, które zachodzą w rzeczywistej dziedzinie problemowej? Nie jest to wcale oczywiste.

Systemy dowodzenia twierdzeń

W logice zdefiniowano szereg **systemów dowodzenia twierdzeń** wprowadzających różne zestawy reguł wnioskowania, a także pewne formuły początkowe, które można w nich stosować (aksjomaty). W tym wykładzie nie będziemy zagłębiać się w konstrukcję tych systemów, jednak musimy znać i rozumieć pewne ich własności.

Reguła wnioskowania jest **poprawna** jeśli można za jej pomocą wywieść fałsz jedynie z niespełnialnego zbioru przesłanek.

Zbiór reguł wnioskowania jest **kompletny** jeśli można z jego pomocą wywieść fałsz z każdego niespełnianego zbioru przesłanek.

System dowodzenia twierdzeń jest **pełny** jeśli jest poprawny i kompletny. W takim systemie ze zbioru przesłanek można wywieść fałsz wtedy i tylko wtedy gdy zbiór ten jest niespełnialny (pod warunkiem, że są to formuły zamknięte).

A więc to co byłoby potrzebne agentowi inteligentnemu, to jest pełny system dowodzenia twierdzeń, z efektywną obliczeniowo procedurą dowodową.

Krótkie podsumowanie — pytania sprawdzające

1. Wyjaśnij dlaczego wnioskowanie agenta o jego świecie powinno być oparte o wynikanie logiczne.

2. Na czym polega dowodzenie twierdzeń i jaką rolę pełnią w nim reguły wnioskowania?

3. Kiedy możemy zastosować dowodzenie twierdzeń w celu wnioskowania o własnościach zachodzących w jakiejś dziedzinie problemowej?

Postać dysjunkcyjna normalna (DNF)

Procedury automatycznego przetwarzania formuł logicznych wymagają zapisywania formuł w pewnych standardowych **postaciach normalnych**.

Formuły atomowe i ich negacje nazywamy **literałami**, np.: $P, \neg Q(x, f(a))$.

Formuła jest w postaci **DNF** (*Disjunctive Normal Form*) jeśli jest alternatywą koniunkcji literałów. Zarówno alternatywę jak i koniunkcję traktujemy tutaj jako spójniki n-arne, dla dowolnego n, niekoniecznie binarne, korzystając z ich łączności.

Dla każdej formuły logicznej istnieje równoważna jej logicznie formuła w postaci DNF. Na przykład, formuła $(\neg P \wedge Q) \vee R$ jest już w postaci DNF, natomiast formułę $\neg P \wedge (Q \vee R)$ można przekształcić do postaci DNF korzystając z rozdzielności koniunkcji względem alternatywy: $(\neg P \wedge Q) \vee (\neg P \wedge R)$.

Przekształcenie formuły do postaci DNF nie jest jednoznaczne; może istnieć wiele formuł w postaci DNF równoważnych danej formule.

Przekształcanie formuł do DNF

Jedną szczególną konstrukcję postaci DNF można otrzymać z zerojedynkowej tabelki definiującej prawdziwość formuły w zależności od prawdziwości wchodzących w jej skład formuł atomowych.

Przykład:

$$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$$

P	Q	$P \Rightarrow Q$	$Q \Rightarrow P$	$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	1	1	1

Wybierając wiersze tabeli, dla których formuła ma wartość prawdy (jedynka w ostatniej kolumnie), konstruujemy formułę w postaci DNF:

$$(\neg P \wedge \neg Q) \vee (P \wedge Q)$$

Postać koniunkcyjna normalna (CNF)

O formule, która jest koniunkcją alternatyw literałów mówimy, że jest w postaci **CNF** (*Conjunctive Normal Form*). Formuły, która są alternatywami literałów nazywamy **klauzulami**. Zatem formuła w postaci CNF jest koniunkcją klauzul. (Dlatego skrót CNF bywa również rozwijany jako *Clausal Normal Form*).

Przykład formuły w postaci CNF: $(P \vee Q \vee \neg R) \wedge (P \vee \neg Q \vee R)$. CNF jest analogiczną, dualną do postaci DNF formą zapisu formuł. Może ona początkowo wydawać się mniej intuicyjna, jest jednak znacznie bardziej przydatna w systemach automatycznego dowodzenia twierdzeń.

Na przykład, postać CNF jest modularna, tzn. gdy chcemy dodać do posiadanej formuły w postaci CNF jakiś nowy fakt w postaci innej formuły CNF, to operacja ta jest trywialna i nie narusza postaci dotychczas posiadanej formuły, w odróżnieniu od DNF.

Przekształcanie formuł do CNF

Rozważmy ponownie przykładową formułę i jej zerojedynkową tabelę prawdy:

$$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$$

P	Q	$P \Rightarrow Q$	$Q \Rightarrow P$	$(P \Rightarrow Q) \wedge (Q \Rightarrow P)$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	1	1	1

Analogicznie do algorytmu konstrukcji postaci kanonicznej DNF formuły, możemy skonstruować postać CNF biorąc wiersze z zerową wartością formuły, i konstruując klauzule eliminujące te wiersze:

$$(\neg P \vee Q) \wedge (P \vee \neg Q)$$

Puste koniunkcje i puste klauzule

Możemy mówić o pojedynczych literałach jako o 1-arnych (unarnych) koniunkcjach klauzul, albo klauzulach (alternatywach literałów). Co więcej, dopuszczamy również klauzule, oraz koniunkcje klauzul, 0-arne, czyli puste.

$$\begin{array}{rcl} p_1 \wedge p_2 \wedge ... \wedge p_n & = & \wedge(p_1, p_2, ..., p_n) \\ p \wedge q & = & \wedge(p, q) \\ p & = & \wedge(p) \\ \sqcup & = & \wedge() \end{array} \qquad \begin{array}{rcl} p_1 \vee p_2 \vee ... \vee p_n & = & \vee(p_1, p_2, ..., p_n) \\ p \vee q & = & \vee(p, q) \\ p & = & \vee(p) \\ \sqcup & = & \vee() \end{array}$$

Zauważmy, że o ile prawdziwość niepustych koniunkcji lub klauzul zależy od prawdziwości ich elementów, to puste koniunkcje oraz klauzule powinny mieć jakąś stałą interpretację logiczną. Przez proste uogólnienie definicji prawdziwości spójników możemy otrzymać fakt, że pusta koniunkcja jest formułą prawdziwą (tautologią), natomiast pusta klauzula jest formułą fałszywą (niespełnialną).

Przy zapisie formuł logicznych w postaci zbiorów lub list, puste koniunkcje i klauzule pojawiają się jako zbiory puste $\{\}$ lub puste listy: () oraz NIL. Dodatkowo stosuje się symbol $\square$ dla zapisu pustej klauzuli w notacji logicznej.

Przykład:

Weźmy formułę $(P \wedge Q)$ zapisaną w postaci zbioru klauzul (jednoliterałowych): $\{P, Q\}$. Dodanie do tego zbioru koniunkcji pustej: $\{P, Q\} \cup \{\}$ odpowiada w notacji logicznej operacji: $(P \wedge Q) \wedge T \equiv (P \wedge Q)$ (gdzie T symbolizuje prawdę). Potwierdza to słuszność interpretacji pustej koniunkcji jako tautologii.

Analogicznie, możemy zapisać klauzulę $(P \vee Q)$ w postaci zbioru literałów: $\{P, Q\}$. Dodanie do tego zbioru klauzuli pustej: $\{P, Q\} \cup \{\}$ odpowiada w notacji logicznej operacji: $(P \vee Q) \vee F \equiv (P \vee Q)$ (gdzie F symbolizuje fałsz). Tak więc również interpretacja pustej klauzuli jako formuły fałszywej jest poprawna.

Przekształcanie formuł logicznych do postaci klauzulowej

Formułę bez zmiennych wolnych możemy przekształcić do postaci **klauzulowej** (inaczej: **prenex**) gdzie wszystkie kwantyfikatory zapisane są przed formułą:

(i) przemianuj zmienne związane kwantyfikatorami na unikalne,
(ii) zastąp koniunkcjami i alternatywami pozostałe spójniki logiczne,
(iii) przesuń negacje do wewnątrz formuł (do symboli predykatów),
(iv) wyodrębnij kwantyfikatory poza formułę,
(v) przekształć formułę do postaci koniunkcyjnej (CNF),
(vi) zastąp wszystkie kwantyfikatory egzystencjalne funkcjami Skolema.

Pierwsze pięć kroków są równoważnościowymi przekształceniami logicznymi (przy zachowaniu właściwej kolejności wyodrębnianych kwantyfikatorów w kroku (iv)). Krok (vi), tzw. **skolemizacja**, polega na zastąpieniu formuł postaci:

$$\forall x_1 \forall x_2 ... \forall x_n \exists y \ \Phi(x_1, x_2, ..., x_n, y)$$

formułą

$$\forall x_1 \forall x_2 ... \forall x_n \ \Phi(x_1, x_2, ..., x_n, f_y(x_1, x_2, ..., x_n))$$

gdzie f_y jest nowo wprowadzonym symbolem funkcyjnym zwanym **funkcją Skolema**. (W przypadku braku kwantyfikatorów $\forall$ będzie to **stała Skolema**.)

Twierdzenie Skolema

Ostatni krok w algorytmie przekształcenia formuły do postaci klauzulowej nie jest równoważnościowym przekształceniem logicznym. To znaczy, dla oryginalnej formuły logicznej Φ, i otrzymanej w wyniku przekształcenia jej postaci klauzulowej Φ', w ogólności $\Phi \not\equiv \Phi'$.

Zachodzi jednak następująca własność, zwana **twierdzeniem Skolema**: dla zamkniętej formuły Φ, jeśli Φ' jest jej postacią klauzulową, to Φ jest spełnialna wtedy i tylko wtedy gdy Φ' jest spełnialna.

Zatem, o ile nie możemy w ogólności posługiwać się postacią klauzulową Φ' zamiast oryginalnej formuły Φ, to jednak możemy posługiwać się tą postacią dla celów dowodzenia spełnialności (lub niespełnialności).

Istnieje niezwykle przydatny schemat wnioskowania, wykorzystujący formuły w postaci klauzulowej, zapisywane często w postaci zbioru (lub listy) klauzul, gdzie poszczególne klauzule są zapisane w postaci zbiorów (lub list) literałów.

Krótkie podsumowanie — pytania sprawdzające

Przekształć do postaci prenex następujące formuły rachunku predykatów:

1. $\forall x\ [(P(x) \Rightarrow Q(x)) \wedge (P(x) \Rightarrow R(x))]$
2. $\forall x\ [(P(x) \wedge Q(x)) \vee (R(x) \wedge S(x))]$
3. $\forall x \exists y\ [P(x) \Rightarrow Q(x, y)]$
4. $\exists x \forall y\ [P(x, y) \Rightarrow Q(A, x)]$
5. $\forall x \exists y\ [P(x, y) \Rightarrow Q(y, f(y))]$

Rezolucja — przypadek klauzul podstawionych

Rezolucja dla dwóch klauzul podstawionych: gdy istnieje wspólny literał mający w dwóch klauzulach przeciwne znaki, to rezolucja tworzy nową klauzulę, zwaną **rezolwentą**, będącą połączeniem oryginalnych dwóch klauzul z wyłączeniem wspólnego literału.

Rozważmy przykład, dla klauzul:

$$P \vee Q(A) \quad \text{oraz} \quad \neg S \vee \neg Q(A)$$

rezolucja utworzy rezolwentę $P \vee \neg S$.

Zauważmy, że dla par klauzul: $(P \vee Q(A)), (\neg S \vee Q(A))$ jak również dla: $(P \vee Q(A)), (\neg S \vee \neg Q(B))$ nie istnieją wspólne literały o przeciwnych znakach, zatem wykonanie rezolucji dla tych par klauzul nie jest możliwe.

Fakt: rezolwenta zawsze wynika logicznie z połączenia (koniunkcji) klauzul wyjściowych, zatem jako reguła wnioskowania rezolucja jest poprawna (*sound*).

Ciekawe przypadki szczególne rezolucji (symbol $\rightsquigarrow$ oznacza tu możliwość wykonania rezolucji i otrzymania wskazanego wyniku):

P	oraz	$\neg P \vee Q$	$\rightsquigarrow$	Q	modus ponens
$P \vee Q$	oraz	$\neg P \vee Q$	$\rightsquigarrow$	Q	mocniejsza wersja
$P \vee Q$	oraz	$\neg P \vee \neg Q$	$\rightsquigarrow$	$P \vee \neg P$	tautologia
$P \vee Q$	oraz	$\neg P \vee \neg Q$	$\rightsquigarrow$	$Q \vee \neg Q$	-"-
P	oraz	$\neg P$	$\rightsquigarrow$	NIL	sprzeczność
$\neg P \vee Q$	oraz	$\neg Q \vee R$	$\rightsquigarrow$	$\neg P \vee R$	przechodniość
$(P \Rightarrow Q)$		$(Q \Rightarrow R)$		$(P \Rightarrow R)$	implikacji

Krótkie podsumowanie — pytania sprawdzające

Dla poniższych zbiorów formuł, zapisz wszystkie możliwe do otrzymania rezolwenty. Jeśli nie jest możliwe wykonanie rezolucji, to wyjaśnij dlaczego nie. Porównaj otrzymane rezolwenty z wnioskami logicznymi, które możesz wyciągnąć intuicyjnie z podanych formuł.
Zwróć uwagę na przecinki, aby prawidłowo odczytać formuły w zbiorach.

1. $\{\ P \vee Q \vee R\ ,\ \neg P \vee \neg Q \vee \neg R\ \}$

2. $\{\ P \vee Q\ ,\ P \vee \neg Q\ ,\ \neg P \vee Q\ \}$

3. $\{\ P \Rightarrow (Q \vee R)\ ,\ \neg Q \wedge \neg R\ \}$

4. $\{\ P \Rightarrow Q\ ,\ R \Rightarrow Q\ ,\ P \vee R\ \}$

Podstawienia zmiennych w formułach

Będziemy rozważali przekształcenia formuł polegające na zastępowaniu zmiennych w formułach innymi wyrażeniami (termami). Ponieważ zmienne w formułach w postaci prenex domyślnie związane są kwantyfikatorami uniwersalnymi, zastępowanie zmiennych innymi termami oznacza branie szczególnych przypadków formuły.

Podstawieniem (*substitution*) nazwiemy zbiór odwzorowań określających termy podstawiane pod poszczególne zmienne. Podstawiane termy nie mogą zawierać zastępowanej zmiennej. Przykład podstawienia:
$s = \{x \mapsto A, y \mapsto f(z)\}$.

Wykonanie podstawienia polega na syntaktycznym zastąpieniu wszystkich wystąpień danej zmiennej w formule związanym z nią termem. Wszystkie zastąpienia wykonywane są jednocześnie, czyli wynikiem wykonania podstawienia $s = \{x \mapsto y, y \mapsto A\}$ na termie $f(x, y)$ będzie term $f(y, A)$.

Zauważ, że w ten sposób nie ma znaczenia w jakiej kolejności zmienne są zastępowane, pomimo iż podstawienie jest zbiorem (nieuporządkowanym).

Złożeniem podstawień s_1 i s_2 (zapisywanym: $s_1 s_2$) nazwiemy podstawienie uzyskane przez zastosowanie podstawienia s_2 do termów z s_1, oraz dopisanie do otrzymanego zbioru wszystkich par z s_2 ze zmiennymi nie występującymi w s_1.

$$\begin{aligned} \Phi s_1 s_2 &= (\Phi s_1) s_2 \\ s_1(s_2 s_3) &= (s_1 s_2) s_3 \end{aligned}$$

Unifikacja

Unifikacją nazywamy takie podstawienie termów pod zmienne w zbiorze formuł, aby sprowadzić wszystkie formuły w zbiorze do identycznych (lub równoważnych logicznie, patrz wyjaśnienie niżej) formuł, czyli zbioru singletonowego.

Unifikator zbioru formuł to jest podstawienie redukujące zbiór do jednoelementowego. Zbiór jest **unifikowalny** jeśli istnieje jego unifikator.

Na przykład: zbiór $\{P(x), P(A)\}$ jest unifikowalny, i jego unifikatorem jest $s = \{x \mapsto A\}$.

Podobnie, zbiór $\{P(x), P(y), P(A)\}$ jest unifikowalny, a jego unifikatorem jest $s = \{x \mapsto A, y \mapsto A\}$.

Zbiór $\{P(A), P(B)\}$ nie jest unifikowalny, podobnie jak zbiór $\{P(A), Q(x)\}$.

Unifikacja (cd.)

Unifikacja jest ogólną procedurą, ale tutaj będziemy wykonywać unifikacje wyłącznie na zbiorach klauzul. Rozważmy poniższe, przykładowe zbiory klauzul:

$$\begin{aligned} \Phi &= \{P \vee Q(x), P \vee Q(A), P \vee Q(y)\} \\ \Psi &= \{P \vee Q(x), P \vee Q(A), P \vee Q(f(y))\} \\ \Omega &= \{P \vee Q(x), P \vee Q(A) \vee Q(y)\} \end{aligned}$$

Zbiór Φ jest unifikowalny, jego unifikatorem jest $s = \{x \mapsto A, y \mapsto A\}$, a zunifikowanym zbiorem jest singletonowy zbiór $\Phi s = \{P \vee Q(A)\}$.

Zbiór Ψ nie jest unifikowalny.

Zbiór Ω jest bardziej skomplikowanym przypadkiem. Stosując czysto **syntaktyczną unifikację**, nie jest on unifikowalny, bo po wykonaniu samego podstawienia formuły nie są identyczne. Jednak stosując **semantyczną unifikację**, jest on unifikowalny, ponieważ po wykonaniu podstawienia formuły są logicznie równoważne. Będziemy dopuszczali unifikację semantyczną z zastosowaniem łączności i przemienności alternatywy.

Najogólniejszy unifikator (mgu)

Najogólniejszym unifikatorem unifikowalnego zbioru formuł, albo **mgu** (*most general unifier*), nazywamy najprostszy (minimalny) unifikator dla tego zbioru.

Dla unifikowalnego zbioru formuł zawsze istnieje jego mgu, a dowolny unifikator dla tego zbioru można otrzymać przez złożenie mgu z jakimś innym podstawieniem. **Algorytm unifikacji** pozwala wyznaczyć mgu zbioru formuł.

Krótkie podsumowanie — pytania sprawdzające

Dla poniższych zbiorów klauzul odpowiedz, czy dany zbiór jest unifikowalny. Jeśli tak, to podaj jego unifikator. Spróbuj podać zarówno mgu, jak i inny unifikator, który nie jest mgu. Jeśli zbiór nie jest unifikowalny, to krótko uzasadnij dlaczego.
Zwróć uwagę na przecinki, aby prawidłowo odczytać formuły w zbiorach.

1. $\{P(x)\ , P(f(x))\}$

2. $\{P(x, y)\ , P(y, x)\}$

3. $\{P(x, y)\ , P(y, f(x))\}$

4. $\{P(x, y)\ , P(y, f(y))\}$

5. $\{P(x, y)\ , P(y, z)\ , P(z, A)\}$

Rezolucja — przypadek ogólny

Rezolucja w ogólnym przypadku: gdy dla dwóch klauzul (zbiorów literałów) $\{L_i\}$ i $\{M_i\}$ istnieją ich podzbiory $\{l_i\}$ i $\{m_i\}$, zwane **literałami kolidującymi**, takie, że zbiór $\{l_i\} \cup \{\neg m_i\}$ daje się zunifikować i s jest jego mgu, wtedy ich rezolwentą jest zbiór $[\{L_i\} - \{l_i\}]s \cup [\{M_i\} - \{m_i\}]s$.

Mogą istnieć różne rezolwenty danych klauzul dla różnych wyborów podzbiorów ich literałów. Na przykład, rozważmy następujące klauzule:

$$P[x, f(A)] \vee P[x, f(y)] \vee Q(y) \quad \text{oraz} \quad \neg P[z, f(A)] \vee \neg Q(z)$$

Wybierając $\{l_i\} = \{P[x, f(A)]\}$ oraz $\{m_i\} = \{\neg P[z, f(A)]\}$ otrzymujemy rezolwentę:

$$P[z, f(y)] \vee \neg Q(z) \vee Q(y)$$

Natomiast wybierając $\{l_i\} = \{P[x, f(A)], P[x, f(y)]\}$ oraz $\{m_i\} = \{\neg P[z, f(A)]\}$ otrzymujemy:

$$Q(A) \vee \neg Q(z)$$

Krótkie podsumowanie — pytania sprawdzające

Dla poniższych zbiorów klauzul, zapisz wszystkie możliwe do otrzymania rezolwenty. Dla każdej rezolwenty zanotuj, z których klauzul została otrzymana, i jakie było zastosowane podstawienie. Jeśli nie jest możliwe wykonanie rezolucji, to wyjaśnij dlaczego nie.
Zwróć uwagę na przecinki, aby prawidłowo odczytać formuły w zbiorach.

1. $\{\neg P(x) \vee Q(x)\ ,\ P(A)\}$

2. $\{\neg P(x) \vee Q(x)\ ,\ \neg Q(x)\}$

3. $\{\neg P(x) \vee Q(x)\ ,\ P(f(x))\ ,\ \neg Q(x)\}$

Rezolucja jako reguła wnioskowania

Rezolucja jest poprawną regułą wnioskowania ponieważ klauzula otrzymana z pary klauzul w wyniku zastosowania rezolucji, wynika z nich logicznie. Jednak nie jest kompletna, to znaczy nie możemy z jej pomocą wygenerować z danej formuły Δ każdego wniosku φ, takiego że $\Delta \vdash \varphi$.

Na przykład, dla $\Delta = \{P, Q\}$ nie możemy rezolucją wywieść formuł $P \vee Q$ ani $P \wedge Q$, a dla $\Delta = \{\forall x R(x)\}$ nie możemy wywieść formuły $\exists x R(x)$.

Jednak jeśli zastosujemy rezolucję w procedurze dowodzenia **nie wprost**, czyli przez zaprzeczenie tezy i wyprowadzenie sprzeczności, reprezentowanej przez klauzulę pustą (oznaczaną $\square$), to możemy udowodnić nią każde twierdzenie. Zatem w tym sensie rezolucja jest kompletna (*refutation complete*).

Rozważmy powyższe przykłady. Dla $\Delta = \{P, Q\}$ zaprzeczeniem formuły $P \vee Q$ są klauzule $\neg P$ i $\neg Q$ i każda z nich pozwala natychmiast wygenerować klauzulę pustą z odpowiednią klauzulą z Δ. Zaprzeczeniem formuły $P \wedge Q$ jest klauzula $\neg P \vee \neg Q$ i możemy uzyskać klauzulę pustą w dwóch krokach rezolucji. Dla $\Delta = \{\forall x R(x)\}$ zaprzeczeniem formuły $\exists x R(x)$ jest $\neg R(y)$, która unifikuje się z klauzulą $R(x)$ otrzymaną z Δ i daje klauzulę pustą w jednym kroku rezolucji.

Dowodzenie twierdzeń oparte na rezolucji

Podstawowy schemat wnioskowania opartego na rezolucji, gdy posiadamy zbiór aksjomatów Δ i chcemy z niego wywieść formułę φ, jest następujący. Łączymy zbiory klauzul otrzymanych z Δ i $\neg\varphi$, i próbujemy wywieść sprzeczność (klauzulę pustą) z otrzymanego łącznego zbioru klauzul, przez zastosowanie rezolucji na kolejnych parach wybranych klauzul. W tym procesie uzyskana w bieżącym kroku rezolucji nowa klauzula zostaje każdorazowo dołączona do głównego zbioru klauzul, i procedura jest powtarzana.

Podstawowy wynik logiki matematycznej tu wykorzystywany jest następujący. Jeśli uruchomimy rezolucję na zbiorze klauzul otrzymanym z niespełnialnej formuły, z jakimś systematycznym algorytmem generowania rezolwent, to w pewnym momencie otrzymamy klauzulę pustą. I na odwrót, jeśli ze zbioru klauzul otrzymanego z jakiejś formuły można procedurą rezolucji wygenerować klauzulę pustą, to ten zbiór klauzul, ale także oryginalna formuła, są niespełnialne. Obejmuje to również klauzule otrzymane w wyniku skolemizacji, a więc jest zarazem potwierdzeniem poprawności tej procedury.

Dowodzenie twierdzeń: przykład

Wiadomo, że:

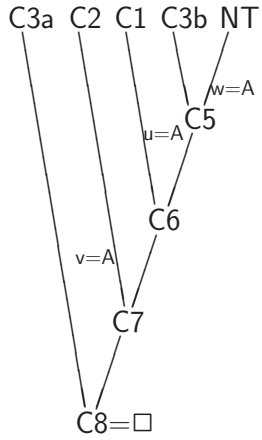
1. Kto potrafi czytać ten jest oświecony. $(\forall x)[R(x) \Rightarrow L(x)]$
2. Delfiny nie są oświecone. $(\forall x)[D(x) \Rightarrow \neg L(x)]$
3. Niektóre delfiny są inteligentne. $(\exists x)[D(x) \wedge I(x)]$

Należy udowodnić twierdzenie:

4. Są tacy inteligentni co nie potrafią czytać. $(\exists x)[I(x) \wedge \neg R(x)]$

Po konwersji do postaci prenex CNF otrzymujemy klauzule:

C1:	$\neg R(u) \vee L(u)$	z pierwszego aksjomatu
C2:	$\neg D(v) \vee \neg L(v)$	z drugiego aksjomatu
C3a:	$D(A)$	z trzeciego aksjomatu, cz.1
C3b:	$I(A)$	z trzeciego aksjomatu, cz.2
NT:	$\neg I(w) \vee R(w)$	z negacji twierdzenia



Z kolejnych kroków rezolucji otrzymujemy:

C5:	$R(A)$	rezolwenta klauzul C3b i NT
C6:	$L(A)$	rezolwenta klauzul C5 i C1
C7:	$\neg D(A)$	rezolwenta klauzul C6 i C2
C8:	$\square$	rezolwenta klauzul C7 i C3a

Dowodzenie twierdzeń: poważniejszy przykład

Rozważmy przykład z matematyki.[2] Chcemy udowodnić, że przekrój dwóch zbiorów zawiera się w dowolnym z nich. Zaczynamy od wypisania aksjomatów, z których rozumowanie będzie musiało korzystać. W tym przypadku są to definicje pojęć przekroju i zawierania się zbiorów.

$$\forall x \forall s \forall t \;\; (x \in s \;\wedge\; x \in t) \Leftrightarrow x \in s \cap t$$
$$\forall s \forall t \;\; (\forall x \; x \in s \Rightarrow x \in t) \Leftrightarrow s \subseteq t$$

Formuła do udowodnienia:

$$\forall s \forall t \;\; s \cap t \subseteq s$$

[2] Przykład zapożyczony z książki Geneseretha i Nilssona „Logical Foundations of Artificial Intelligence".

Po przetworzeniu do postaci klauzul otrzymujemy:

1.	$\{x \notin s, x \notin t, x \in s \cap t\}$	z definicji przekroju
2.	$\{x \notin s \cap t, x \in s\}$	z definicji przekroju
3.	$\{x \notin s \cap t, x \in t\}$	z definicji przekroju
4.	$\{F(s,t) \in s, s \subseteq t\}$	z definicji zawierania się
5.	$\{F(s,t) \notin t, s \subseteq t\}$	z definicji zawierania się
6.	$\{A \cap B \not\subseteq A\}$	z zaprzeczenia tezy

Zauważmy funkcje Skolema w klauzulach 4 i 5, oraz stałe Skolema w klauzuli 6. Dalej następuje wywód prowadzący dosyć prosto do klauzuli pustej.

7.	$\{F(A \cap B, A) \in A \cap B\}$	z klauzul 4. i 6.
8.	$\{F(A \cap B, A) \notin A\}$	z klauzul 5. i 6.
9.	$\{F(A \cap B, A) \in A\}$	z klauzul 2. i 7.
10.	$\{\}$	z klauzul 8. i 9.

To koniec dowodu. Cel osiągnięty. Trochę trudno poczuć satysfakcję jaka zwykle towarzyszy osiągnięciu tradycyjnego dowodu matematycznego. Również aby prześledzić rozumowanie i np. je sprawdzić, trzeba się trochę napracować, aczkolwiek w przypadku tego dowodu jest to jeszcze względnie proste.

Krótkie podsumowanie — pytania sprawdzające

Dla podanych poniżej zbiorów aksjomatów Δ i formuły φ, spróbuj udowodnić $\Delta \vdash \varphi$ metodą rezolucji nie wprost. Zacznij od zaprzeczenia formuły celowej, następnie z otrzymanego zaprzeczenia i zbioru aksjomatów utwórz podstawowy zbiór klauzul.

1. $\Delta = \{\forall x(\mathsf{Lubi}(x, \mathsf{Wino}) \Rightarrow \mathsf{Lubi}(\mathsf{Rychu}, x)), \mathsf{Lubi}(\mathsf{Zdzich}, \mathsf{Wino})\}$
   $\varphi = \mathsf{Lubi}(\mathsf{Rychu}, \mathsf{Zdzich})$

2. $\Delta = \{\forall x(\mathsf{Lubi}(x, \mathsf{Rychu}) \Rightarrow \mathsf{Lubi}(\mathsf{Rychu}, x)), \neg\mathsf{Lubi}(\mathsf{zona}(\mathsf{Zdzich}), \mathsf{Rychu})\}$
   $\varphi = \mathsf{Lubi}(\mathsf{Rychu}, \mathsf{zona}(\mathsf{Zdzich}))$

3. $\Delta = \{\forall x(\mathsf{Lubi}(x, \mathsf{Wino}) \Rightarrow \mathsf{Lubi}(\mathsf{Rychu}, x)), \mathsf{Lubi}(\mathsf{Zdzich}, \mathsf{Wino})\}$
   $\varphi = (\mathsf{Lubi}(\mathsf{Rychu}, \mathsf{Zdzich}) \vee \mathsf{Lubi}(\mathsf{Rychu}, \mathsf{zona}(\mathsf{Zdzich}))$

4. $\Delta = \{\forall x(\mathsf{Lubi}(x, \mathsf{Wino}) \Rightarrow \mathsf{Lubi}(\mathsf{Rychu}, x)), \mathsf{Lubi}(\mathsf{Zdzich}, \mathsf{Wino})\}$
   $\varphi = (\mathsf{Lubi}(\mathsf{Rychu}, \mathsf{Zdzich}) \wedge \mathsf{Lubi}(\mathsf{Rychu}, \mathsf{zona}(\mathsf{Zdzich}))$

Inżynieria wiedzy

Przedstawiony formalizm logiki predykatów pierwszego rzędu, wraz z rezolucją jako metodą dowodzenia twierdzeń nie wprost, pozwala na budowę inteligentnych agentów efektywnie rozwiązujących stawiane im problemy. Budowa takiego agenta wymaga konstrukcji reprezentacji, którą można sformułować w postaci następującego procesu, zwanego **inżynierią wiedzy**:

identyfikacja problemu
: określenie zakresu pytań, na które agent będzie musiał znajdować odpowiedzi, rodzaju faktów, którymi będzie mógł się posługiwać, itp.; na przykład, w odniesieniu do świata wumpusa, musimy określić, czy agent ma umieć planować działania, czy np. tylko tworzyć reprezentację stanu świata rozpoznanego w dotychczasowych działaniach?

pozyskanie wiedzy
: twórca oprogramowania agenta (inżynier wiedzy) może nie rozumieć wszystkich niuansów opisywanego świata, i musi współpracować z ekspertami aby pozyskać całą niezbędną wiedzę

definicja słownika reprezentacji
: pojęcia i obiekty z dziedziny problemowej muszą zostać opisane formułami logicznymi; konieczne jest zdefiniowanie słownika predykatów i termów, tzn. funkcji termowych oraz stałych; ten etap może się okazać kluczowy dla zdolności do efektywnego rozwiązywania problemów, np. w świecie wumpusa, czy zapadliny lepiej przedstawić jako obiekty, czy własności miejsc

kodowanie wiedzy ogólnej
: kodowanie aksjomatów zawierających ogólną wiedzę o dziedzinie problemowej, regułach rządzących światem, istniejących heurystykach, itp.

kodowanie wiedzy szczególnej
: zapis konkretnego problemu do rozwiązania przez agenta, w tym zadanych mu faktów o konkretnych obiektach, oraz postawionego zadania jako pytania do odpowiedzenia lub, ogólniej, twierdzenia do udowodnienia

przedstawienie zapytań do urządzenia wnioskującego
: uruchomienie procedury dowodzenia na skonstruowanej bazie wiedzy + faktach

debugowanie bazy wiedzy

niestety, podobnie jak w przypadku zwykłych programów, rzadko kiedy skonstruowany system będzie od razu poprawnie działał; mogą zdarzyć się takie problemy, jak brak kluczowych aksjomatów, albo aksjomaty nieprecyzyjnie sformułowane, które pozwalają na udowodnienie zbyt mocnych twierdzeń

Algorytmy pomocnicze: relacja równości

Jedną ze szczególnych relacji występujących w formułach logicznych jest relacja równości (identyczności) termów.

Przykład:
$\Delta = \{=(\text{żona}(\text{Zdzich}), \text{Gabrysia}), \text{Posiada}(\text{żona}(\text{Zdzich}), \text{alfa-8c})\}$.
Czy to znaczy, że Gabrysia posiada Alfę 8c Competizione?
Czy możemy to udowodnić metodą rezolucji?
$\text{Posiada}(\text{Gabrysia}, \text{alfa-8c})$?

Niestety, nie. Procedura dowodowa rezolucji nie traktuje predykatu równości w żaden szczególny sposób i nie wykorzysta posiadanej informacji o identyczności termów. Aby dowód w powyższym przykładzie był możliwy musielibyśmy sformułować dodatkowy aksjomat równości:

$$\forall x, y, z\ [\text{Posiada}(x, y) \wedge =(x, z) \Rightarrow \text{Posiada}(z, y)]$$

Za pomocą sformułowanego powyżej aksjomatu równości można udowodnić, że Gabrysia posiada Alfę, jak również podobne fakty o posiadaniu dla innych posiadaczy określanych jawnie lub niejawnie. Jednak aby móc podobne wnioskowanie rozciągnąć na równoważność przedmiotu posiadania, niezbędny jest jeszcze inny aksjomat:

$$\forall x, y, z\ [\text{Posiada}(x, y) \wedge =(y, z) \Rightarrow \text{Posiada}(x, z)]$$

Co gorsza, aby system mógł sprawnie posługiwać się znanymi faktami tożsamości termów w odniesieniu do wszystkich relacji, analogiczne aksjomaty równości należałoby napisać dla wszystkich symboli predykatów. Niestety, w języku logiki pierwszego rzędu nie można tego wyrazić jedną wspólną formułą.

Alternatywnym rozwiązaniem jest wbudowanie obsługi równości termów w procedurę dowodzenia. Istnieje kilka rozwiązań tego problemu: reguła redukcji formuł ze względu na relację równości zwana **demodulacją**, uogólniona reguła rezolucji uwzględniająca relację równości zwana **paramodulacją**, oraz wbudowanie relacji równości w procedurę unifikacji.

Algorytmy pomocnicze: odzyskiwanie odpowiedzi z drzewa rezolucji

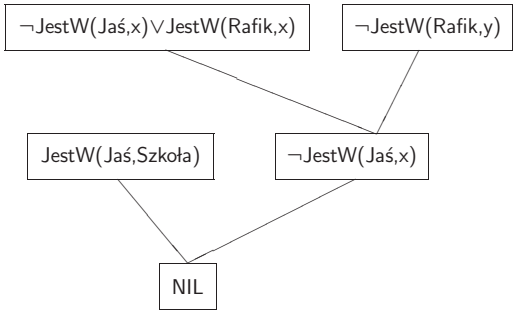
Rozważmy następujący przykład, wiemy że:

Gdzie jest Jaś, tam jest Rafik.	$(\forall x)[\text{JestW}(\text{Jaś}, x) \Rightarrow \text{JestW}(\text{Rafik}, x)]$
Jaś jest w szkole.	$\text{JestW}(\text{Jaś}, \text{Szkoła})$

Chcemy znaleźć odpowiedź na pytanie:

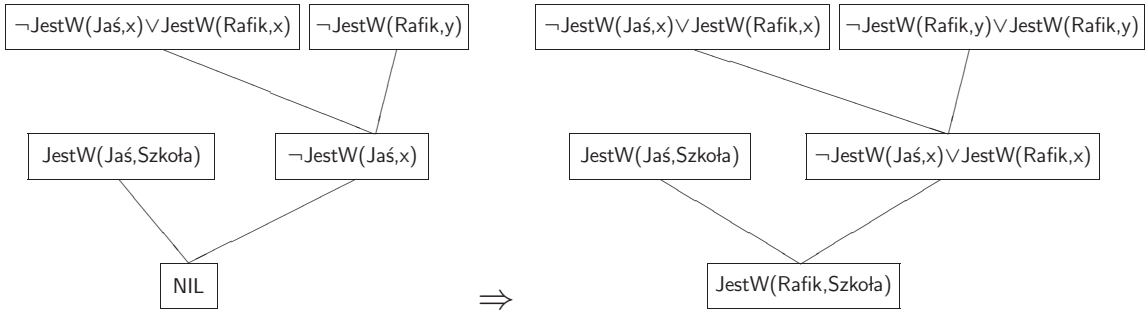
Gdzie jest Rafik?	$(\exists x)[\text{JestW}(\text{Rafik}, x)]$

Formuła logiczna odpowiadająca oryginalnemu pytaniu różni się nieco od niego, ale dowód znajduje się łatwo:



Niestety, nie daje on odpowiedzi na pierwotnie postawione pytanie.

Odzyskiwanie odpowiedzi z drzewa rezolucji (c.d.)



- Podstawowa procedura zamienia dowód nie wprost na kompletny dowód tezy wprost.
- Jeśli twierdzenie zawiera alternatywy (po zaprzeczeniu stają się koniunkcjami) to w wyniku tej procedury otrzymujemy złożoną formułę, która może być trudna do interpretacji.
- Jeśli twierdzenie zawiera kwantyfikator ogólny to po zaprzeczeniu pojawiają się w niej stałe lub funkcje Skolema, które lądują w odpowiedzi, ale mogą być zamienione na zmienne kwantyfikowane uniwersalnie.

Algorytmy pomocnicze: strategie przyspieszające rezolucję

W dowodzeniu twierdzeń z wykorzystaniem rezolucji w procedurze dowodowej nie wprost, dążymy do wygenerowania klauzuli pustej, wskazującej na sprzeczność. Aby mieć pewność, że taką klauzulę wygenerujemy, zakładając, że jest to w ogóle możliwe, musimy generować rezolwenty w jakiś systematyczny sposób, na przykład realizując przeszukiwanie wszerz. Jednak przy większych bazach danych, ten sposób może prowadzić do generowania bardzo wielu wniosków, z których większość może nie mieć nic wspólnego z dowodzonym twierdzeniem.

Poszukuje się zatem **strategii przyspieszających**, które pozwoliłyby odciąć i nie generować niektórych rezolwent. Strategie takie mogą być **pełne**, tzn. dające gwarancję znalezienia rozwiązania (fałszu) jeśli to tylko możliwe, albo **niepełne**, czyli nie dające takiej gwarancji (ale typowo znacznie skuteczniejsze).

Strategie przyspieszające:

- preferencja pojedynczych literałów (normalnie niepełna, ale pełna jeśli jest traktowana tylko jako preferencja)
- strategia zbioru podtrzymania (*set of support*): tylko rezolucje z klauzulami z pewnego zbioru, początkowo uzyskanego z zaprzeczonej tezy (pełna)
- rezolucja wejściowa (*input resolution*) zezwala tylko na rezolucje z użyciem klauzuli wejściowej (pełna tylko w niektórych przypadkach, np. dla hornowskich baz danych)
- rezolucja liniowa (niepełna)
- eliminacja powtórzeń i specjalizacji (pełna)

Nierozstrzygalność rachunku predykatów

Rachunek predykatów wydaje się językiem reprezentacji dobrze nadającym się do wyrażania faktów i wnioskowania w systemach sztucznej inteligencji. Należy jednak zdawać sobie sprawę z pewnych jego ograniczeń, które zawężają jego użyteczność praktyczną.

Twierdzenie Churcha (1936, o nierozstrzygalności rachunku predykatów): nie istnieje procedura pozwalająca w ogólnym przypadku sprawdzać prawdziwości formuł rachunku predykatów. Mówimy, że rachunek predykatów jest **nierozstrzygalny** (*undecidable*).

Ta własność w istotny sposób ogranicza możliwości wnioskowania o faktach wyrażanych w języku predykatów. Co prawda istnieje szereg klas formuł, dla których procedura decyzyjna istnieje. Poza tym rachunek predykatów ma własność **półrozstrzygalności** (*semidecidability*), co oznacza, że istnieje procedura pozwalająca stwierdzić niespełnialność dowolnej formuły niespełnialnej w skończonej liczbie kroków. Niestety, dla formuł, które nie są niespełnialne, ta procedura może nigdy się nie zatrzymać.

Niezupełność w rachunku predykatów

Ktoś mógłby myśleć, że nierozstrzygalność rachunku predykatów można pokonać, korzystając z półrozstrzygalności. Chcąc udowodnić formułę φ ze zbioru aksjomatów Δ, uruchamiamy równolegle dwa dowody: $\Delta \vdash \varphi$ i $\Delta \vdash \neg\varphi$. Na mocy półrozstrzygalności, przynajmniej jeden z tych dowodów powinien zakończyć się powodzeniem. Niestety, to również może się nie udać.

Twierdzenie Gödla (1931, o niezupełności): w rachunku predykatów można sformułować teorie **niezupełne**, czyli takie, w których istnieją formuły zamknięte, których nie można wywieść, ani nie można wywieść ich zaprzeczenia. Co więcej, takie teorie są dość proste, na przykład taką teorią jest teoria liczb naturalnych, generowana opisującym je zbiorem aksjomatów.

Teorię $\mathcal{T}$ nazywamy **rozstrzygalną** jeśli istnieje algorytm pozwalający dla dowolnej formuły zamkniętej φ stwierdzić, czy $\varphi \in \mathcal{T}$, czy $\varphi \notin \mathcal{T}$. Teorie niezupełne są więc w oczywisty sposób nierozstrzygalne.

Twierdzenie Gödla ma taki skutek, że jeśli po pewnej liczbie kroków dowodu $\Delta \vdash \varphi$ (i, być może, równoległego dowodu $\Delta \vdash \neg\varphi$), nadal nie ma rozwiązania, to nie możemy mieć pewności, czy dowód jednak się zakończy (przynajmniej jeden z nich), czy mamy do czynienia z teorią niezupełną.

Zmiany zachodzące w czasie

Rachunek predykatów dobrze spisuje się jako język reprezentacji dla dziedzin statycznych, gdzie nic się nie dzieje, i wszystko co jest prawdziwe, pozostaje takie na zawsze. Świat realny niestety taki nie jest.

Na przykład, jeśli formuła: JestW(Jaś, Szkoła) poprawnie opisuje stan bieżący jakiegoś powszedniego dnia przed południem, to niestety, musimy się liczyć z tym, że Jaś pójdzie w pewnym momencie do domu. Jeśli aksjomatyka dobrze opisuje skutki działań agentów, to system mógłby nawet wywnioskować nowy fakt: JestW(Jaś, Dom). Niestety, wtedy w bazie danych powstanie sprzeczność, która dla systemu logicznego jest katastrofą. System dowodzenia zawierający fałsz w swoich aksjomatach może udowodnić dowolne twierdzenie!

Ćwiczenie: załóżmy, że w zbiorze aksjomatów Δ istnieją, między innymi, dwie klauzule: P i $\neg P$. Przedstaw dowód dowolnej formuły Q. Wskazówka: najpierw udowodnij, że $P \vee \neg P \vee Q$ jest tautologią (zdaniem zawsze prawdziwym) dla dowolnych P i Q. Można to sprawdzić wykonując dowód $\models (P \vee \neg P \vee Q)$, czyli dowodząc tezy z pustym zbiorem aksjomatów. Następnie dodaj tak udowodnioną tautologię do bazy Δ i przeprowadź dowód Q.

Logiki czasowe

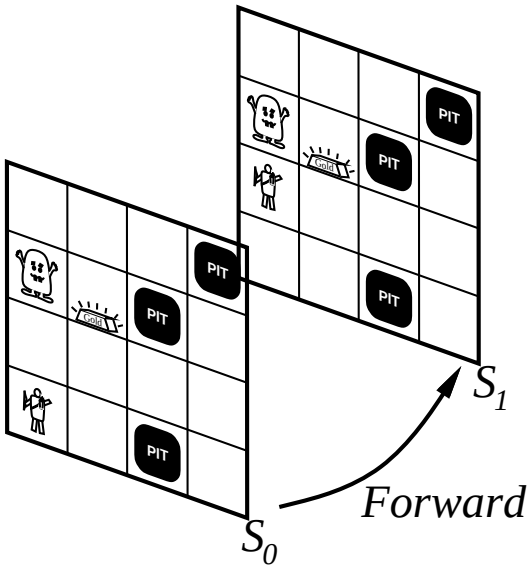
Dla rozwiązania problemu reprezentacji zmian stworzono wiele specjalistycznych teorii logicznych, zwanych **logikami czasowymi** (*temporal logics*). Zwykłe fakty wyrażane w tych logikach zachodzą w określonych momentach czasowych. Natomiast czas, jego własności, i specjalne reguły wnioskowania dotyczące jego upływu, są w logikach czasowych wbudowane w teorię (zamiast być reprezentowane jawnie, na równi z innymi własnościami świata).

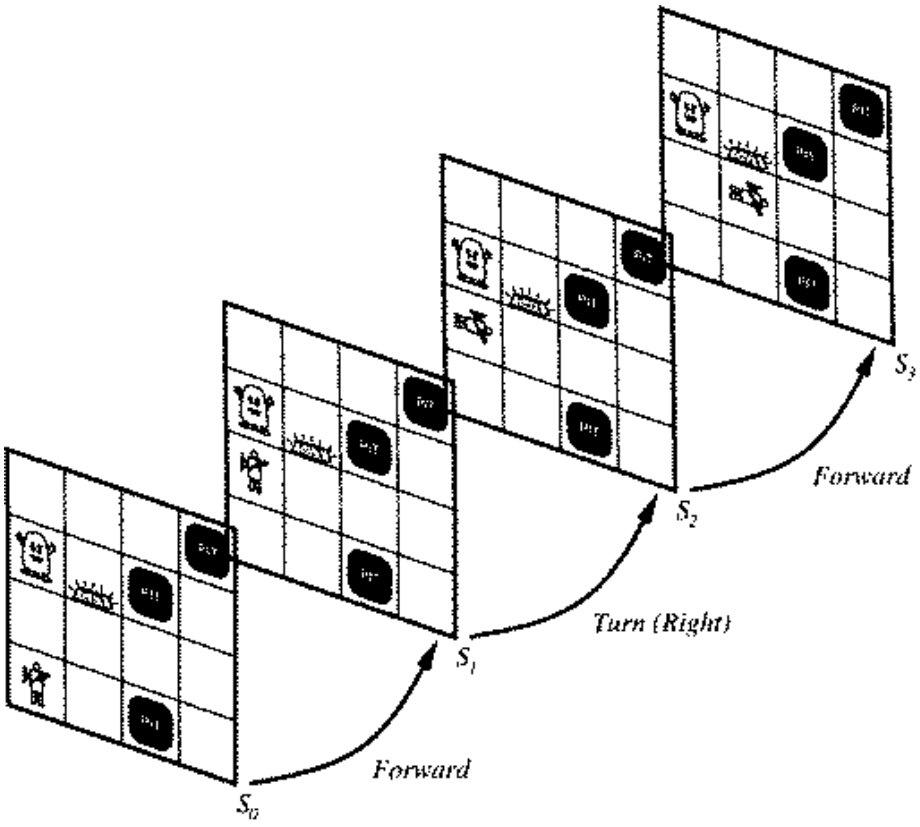
Jedną z głównych kwestii, różniących te teorie, jest sama reprezentacja czasu. Może on być dyskretny lub ciągły, może być określany w postaci punktów lub przedziałów, może być ograniczony lub nieograniczony, itp. Co więcej, czas może być pojęciem liniowym, lub rozgałęzionym. Zwykle powinien jednak być uporządkowany, choć istnieją kołowe reprezentacje czasu.

Dla każdej z takich logik czasowych, aby dało się efektywnie wnioskować o tworzonych formułach, reprezentujących zjawiska, z którymi ma do czynienia inteligentny agent, musi istnieć procedura dowodowa. Konstrukcja takiej procedury może opierać się na rzutowaniu danej teorii do logiki predykatów pierwszego rzędu.

Rachunek sytuacji

Alternatywą do logik czasowych jest bezpośredni zapis momentów czasowych w języku reprezentacji. Przykładem takiego podejścia jest **rachunek sytuacji**:
$At(Agent, [1,1], S_0) \wedge At(Agent, [1,2], S_1) \wedge S_1 = Result(Forward, S_0)$





Rachunek sytuacji (cd.)

Rachunek sytuacji wykorzystuje pojęcia: **sytuacji**, **akcji**, i **fluentów**:

sytuacje: sytuacją jest stan początkowy s_0, i dla każdej sytuacji s i akcji a sytuacją jest również $Result(a, s)$; sytuacje odpowiadają sekwencjom akcji i w ten sposób są różne od stanów, tzn. agent może znaleźć się w danym stanie poprzez różne sytuacje,

fluenty: funkcje i relacje, które podlegają zmianom w czasie nazywane są fluentami i posiadają argument sytuacji, typowo jest to ostatni argument,

aksjomaty dopuszczalności akcji: opisują warunki stosowalności akcji, np. dla akcji $Shoot$: $Have(Agent, Arrow, s) \Rightarrow Poss(Shoot, s)$

aksjomaty następstwa akcji: opisują następstwa akcji dla wszystkich fluentów, np. dla akcji $Grab$ aksjomat powinien stwierdzać, że po prawidłowym wykonaniu tej akcji agent będzie trzymał to co podniósł; należy jednak pamiętać również o sytuacjach, kiedy fluent nie uległ zmianie w wyniku wykonania jakiejś akcji:

$$Poss(a, s) \Rightarrow \\ (Holding(Agent, g, Result(a, s)) \Leftrightarrow \\ a = Grab(g) \vee (Holding(Agent, g, s) \wedge a \neq Release(g))).$$

aksjomaty unikalności akcji: ze względu na obecność klauzul różności akcji w aksjomatach następstwa, musimy zapewnić mechanizm pozwalający agentowi stwierdzać takie fakty, na przykład przez aksjomaty unikalności akcji; dla każdej pary symboli akcji A_i i A_j musimy zapisać aksjomat (na pozór oczywisty) $A_i \neq A_j$; ponadto dla akcji z parametrami musimy zapisać też:

$$A_i(x_1, ..., x_n) = A_j(y_1, ..., y_n) \Leftrightarrow x_1 = y_1 \wedge ... \wedge x_n = y_n$$

Przykład z małpą i bananami — aksjomatyzacja

• wiedza ogólna o świecie i operatorach (częściowa i uproszczona):

A1: $\forall p \forall p_1 \forall s$ [At(box, p, s) $\Rightarrow$ At(box, p, goto(p_1, s))]
A2: $\forall p \forall p_1 \forall s$ [At(bananas, p, s) $\Rightarrow$ At(bananas, p, goto(p_1, s))]
A3: $\forall p \forall s$ [At(monkey, p, goto(p, s))]
A4: $\forall p \forall p_1 \forall s$ [At(box, p, s) $\wedge$ At(monkey, p, s) $\Rightarrow$ At(box, p_1, move(box, p, p_1, s))]
A5: $\forall p \forall p_1 \forall p_2 \forall s$ [At(bananas, p, s) $\Rightarrow$ At(bananas, p, move(box, p_1, p_2, s))]
A6: $\forall p \forall p_1 \forall s$ [At(monkey, p, s) $\Rightarrow$ At(monkey, p_1, move(box, p, p_1, s))]
A7: $\forall s$ [Under(box, bananas, s) $\Rightarrow$ Under(box, bananas, climb(box, s))]
A8: $\forall p \forall s$ [At(box, p, s) $\wedge$ At(monkey, p, s) $\Rightarrow$ On(monkey, box, climb(box, s))]
A9: $\forall s$ [Under(box, bananas, s) $\wedge$ On(monkey, box, s) $\Rightarrow$ Havebananas(grab(bananas, s))]
A10: $\forall p \forall s$ [At(box, p, s) $\wedge$ At(bananas, p, s) $\Rightarrow$ Under(box, bananas, s)]

• dane zadania:

A11: [At(monkey, P_1, S_0) $\wedge$ At(box, P_2, S_0) $\wedge$ At(bananas, P_3, S_0)]

• teza do udowodnienia:

$\exists s$(Havebananas(s))

[2] Przedstawione tutaj rozwiązanie problemu małpy i bananów wzorowane jest na przykładzie z książki Philipa C. Jacksona Jr.'a: „Artificial Intelligence".

Przykład z małpą i bananami — zbiór klauzul

A1: $\{\neg$At(box, p, s_1), At(box, p, goto(p_1, s_1))$\}$
A2: $\{\neg$At(bananas, q, s_2), At(bananas, q, goto(q_1, s_2))$\}$
A3: $\{$At(monkey, r, goto(r, s_3))$\}$
A4: $\{\neg$At(box, u, s_4), $\neg$At(monkey, u, s_4), At(box, u_1, move(box, u, u_1, s_4))$\}$
A5: $\{\neg$At(bananas, t, s_5), At(bananas, t, move(box, t_2, t_3, s_5))$\}$
A6: $\{\neg$At(monkey, v_1, s_6), At(monkey, v_2, move(box, v_1, v_2, s_6))$\}$
A7: $\{\neg$Under(box, bananas, s_7), Under(box, bananas, climb(box, s_7))$\}$
A8: $\{\neg$At(monkey, w, s_8), $\neg$At(box, w, s_8), On(monkey, box, climb(box, s_8))$\}$
A9: $\{\neg$Under(box, bananas, s_9), $\neg$On(monkey, bananas, s_9), Havebananas(grab(bananas, s_9))$\}$
A10: $\{\neg$At(box, p, s_{10}), $\neg$At(bananas, p, s_{10}), Under(box, bananas, s_{10})$\}$
A11a: $\{$At(monkey, P_1, S_0)$\}$
A11b: $\{$At(box, P_2, S_0)$\}$
A11c: $\{$At(bananas, P_3, S_0)$\}$
NT: $\{\neg$Havebananas(z)$\}$

Przykład z małpą i bananami — dowód

C1(A1,A11b) {At(box, P_2, goto(p_1, S_0))}

C2(C1,A4) {¬At(bananas, P_2, goto(p_1, S_0)),
At(box, u_1, move(box, P_2, u_1, goto(p_1, S_0)))}

C3(C2,A3) {At(box, u_1, move(box, P_2, u_1, goto(P_2, S_0)))}

C4(C3,A10) {¬At(bananas, u_1, move(box, P_2, u_1, goto(P_2, S_0))),
Under(box, bananas, move(box, P_2, u_1, goto(P_2, S_0)))}

C5(A2,A11c) {At(bananas, P_3, goto(q_1, S_0))}

C6(C5,A5) {At(bananas, P_3, move(box, t_2, t_3, goto(q_1, S_0)))}

C7(C6,C4) {Under(box, bananas, move(box, P_2, P_3, goto(P_2, S_0)))}

C8(C7,A7) {Under(box, bananas, climb(box, move(box, P_2, P_3, goto(P_2, S_0))))}

C9(A3,A6) {At(monkey, v_2, move(box, r, v_2, goto(r, r_1)))}

C10(C9,A8) {At(box, v_2, move(box, r, v_2, goto(r, r_1))),
On(monkey, box, climb(box, move(box, r, r_2, goto(r, r_1))))}

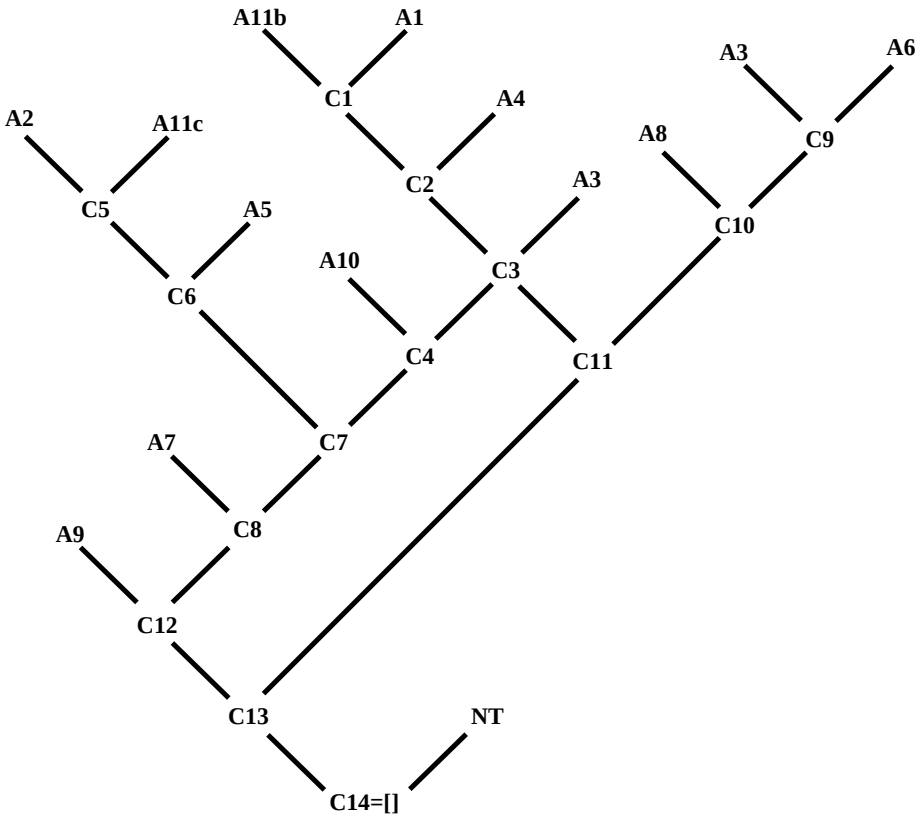
C11(C10,C3) {On(monkey, box, climb(box, move(box, P_2, u_1, goto(P_2, S_0))))}

C12(C8,A9) {¬On(monkey, box, climb(box, move(box, P_2, P_3, goto(P_2, S_0)))),
Havebananas(grab(bananas,
climb(box, move(box, P_2, P_3, goto(P_2, S_0))))))}

C13(C11,C12) {Havebananas(grab(bananas,
climb(box, move(box, P_2, P_3, goto(P_2, S_0))))))}

C14(C13,NT) {}

Przykład z małpą i bananami — drzewo dowodu



Frame problem

Poprawna reprezentacja zagadnienia wymaga jawnego zapisania efektów działań agenta w środowisku, jak również zmian wywołanych przez inne czynniki (np. deszcz). Jednak, jak widzieliśmy w przykładach z wumpusem oraz małpą i bananami, konieczne jest również sformułowanie aksjomatów niezmienniczości, pozwalające na wnioskowanie o braku zmiany:

$$\forall a, x, s\ Holding(x, s) \wedge (a \neq Release) \Rightarrow Holding(x, Result(a, s))$$
$$\forall a, x, s\ \neg Holding(x, s) \wedge (a \neq Grab) \Rightarrow \neg Holding(x, Result(a, s))$$

Niestety, w świecie bardziej złożonym niż świat wumpusa, fluentów będzie bardzo wiele, i aksjomatyka musi opisywać ich zmiany oraz niezmienność, zarówno jako bezpośrednie, jak i uboczne, efekty wykonywanych akcji.

Te aksjomaty, zwane **aksjomatami tła** (*frame axioms*) trudno jest wyrazić w sposób ogólny, i w znacznym stopniu komplikują pierwotny opis świata.

Ponieważ w czasie pracy agent musi odpowiadać sobie na wiele pytań, prowadząc dowody logiczne, mnogość aksjomatów powoduje gwałtowne powiększanie się jego bazy danych, i w efekcie może doprowadzić do kompletnego paraliżu.

Krótkie podsumowanie — pytania sprawdzające

1. Opracuj opartą na rachunku sytuacji reprezentację dla świata wumpusa, przedstawionego na początku tego wykładu.

Problemy z brakiem informacji

Przedstawione dotychczas metody oparte na logice zakładały, że wszystkie informacje niezbędne do przeprowadzenia wywodów logicznych są agentowi dostępne. Niestety, nie jest to realistyczne założenie.

Jednym z problemów jest problem niepełnej informacji. Agent może nie mieć pełnej informacji o problemie, pozwalającej mu na kategoryczne wyciąganie wniosków, lecz może również mieć informacje częściowe, np.:

- fakty „typowe",
- fakty „możliwe",
- fakty „prawdopodobne",
- wyjątki od faktów ogólnie słusznych.

Posiadanie takich informacji często jest kluczowe dla podejmowania właściwych decyzji, lecz klasyczna logika predykatów nie potrafi robić z nich użytku.

Innym problemem jest niepewność informacji. Agent może mieć dane pochodzące z różnych nie w pełni wiarygodnych źródeł. W braku pewnej informacji powinien je wykorzystać, prowadząc wywody opierając się na możliwie najlepszych danych, oraz oszacowując wiarygodność otrzymanych wniosków logicznych. Ponownie, logika klasyczna nie dostarcza takich narzędzi.

Logika zdrowego rozsądku

Zastanówmy się, jakie informacje człowiek wie na pewno, podejmując decyzje w codziennym życiu. Wstając rano, ma zamiar pojechać do pracy. Ale jeśli jest jakaś awaria komunikacji miejskiej, to powinien wstać dużo wcześniej, i najpierw sprawdzić, czy kursują autobusy. Dzień wcześniej, zakupił produkty, aby przyrządzić z nich śniadanie. Ale czy wie na pewno, że jego sałatka śniadaniowa jest nadal w lodówce, czy nie zepsuła się, czy ktoś jej nie wykradł, itp.

Wniosek: logicznie funkcjonujący agent potrzebuje pewnych informacji do swego działania, i prędzej lub później zostanie sparaliżowany stuprocentową poprawnością swego mechanizmu wnioskowania. W świecie rzeczywistym nigdy nie będzie w stanie odważyć się na jakiekolwiek działanie, dopóki nie będzie miał pełnej informacji o otaczającym go świecie.

Jednak ludzie potrafią sprawnie poruszać się w świecie pełnym informacji niepewnej i niepełnej, faktów domyślnych i wyjątków. Jak to robią? Musimy uznać, że w swoim wnioskowaniu ludzie posługują się nieco inną logiką, niż rygorystyczna logika matematyczna. Możnaby ogólnie nazwać ten hipotetyczny mechanizm wnioskowania **logiką zdrowego rozsądku** (*common sense reasoning*).

Logiki niemonotoniczne

Część winy za problemy z wnioskowaniem przy użyciu logiki klasycznej ponosi jej własność określana jako **monotoniczność**. W logice klasycznej, im więcej wiemy, tym więcej możemy wywieść stosując wnioskowanie.

Człowiek stosuje inny model wnioskowania, o wiele bardziej elastyczny, wykorzystujący informację typową, domyślną, możliwą, a nawet brak informacji. Ten rodzaj wnioskowania wydaje się nie mieć własności monotoniczności.

Na przykład, o ile w braku dobrej informacji o sytuacji człowiek byłby gotów wyciągnąć pewne wnioski i podejmować decyzje (pochopne), to po zdobyciu pełniejszej informacji może już nie być w stanie wymyślić dobrego rozwiązania problemu.[3]

Stąd różne modele wnioskowania, zmierzające do pokonania tych problemów, i bardziej zbliżone do elastycznego modelu wnioskowania człowieka, określa się wspólnym mianem **logik niemonotonicznych**.

[3] Rozwiązanie, które wypracował wcześniej, w braku informacji, było błędne, ale może było lepsze niż brak jakiegokolwiek działania. Chociaż niekoniecznie.

Logiki niemonotoniczne — przykład

Wyzwanie Minsky-ego: opracowanie systemu, który pozwoliłby prawidłowo opisać ogólnie znany fakt, że ptaki potrafią fruwać.

$$\forall x[\mathsf{bird}(x) \rightarrow \mathsf{canfly}(x)]$$

Aby uwzględnić wyjątki, np. strusie, trzeba każdorazowo modyfikować poprzednią formułę.

$$\forall x[\mathsf{bird}(x) \wedge \neg\mathsf{ostrich}(x) \rightarrow \mathsf{canfly}(x)]$$

Wyjątków jest więcej: ptaki skąpane w rozlanej ropie naftowej, ptaki bez skrzydeł, chore ptaki, martwe ptaki, namalowane ptaki, abstrakcyjne ptaki, ...

Pomysł: wprowadzamy operator modalny **M**:

$$\forall x[\mathsf{bird}(x) \wedge \mathbf{M}\ \mathsf{canfly}(x) \rightarrow \mathsf{canfly}(x)]$$

Teraz wyjątki możemy wprowadzać modularnie:

$$\forall x[\mathsf{ostrich}(x) \rightarrow \neg\mathsf{canfly}(x)]$$

Dla następującego zbioru faktów:

$$\Delta = \{\mathsf{bird}(\mathit{Tweety}), \mathsf{bird}(\mathit{Sam}), \mathsf{ostrich}(\mathit{Sam})\}$$

możemy wywieść: $\neg\mathsf{canfly}(\mathit{Sam})$
zatem nie powinno nam się udać wyprowadzić:

$$\mathbf{M}\ \mathsf{canfly}(\mathit{Sam}) \qquad \text{ani} \qquad \mathsf{canfly}(\mathit{Sam})$$

Jednak przy użyciu normalnych procedur nie możemy udowodnić zdolności do latania *Tweety*:

$$\mathbf{M}\ \mathsf{canfly}(\mathit{Tweety}), \mathsf{canfly}(\mathit{Tweety})$$

W tym celu niezbędna jest procedura dowodowa zdolna do efektywnego (i automatycznego) dowodzenia twierdzeń w języku predykatów rozszerzonym o operator modalny **M**, zgodna z następującą regułą wnioskowania:

$$\frac{\mathsf{Not}(\vdash \neg p)}{\mathbf{M}\ p}$$

Logiki niemonotoniczne — jaka procedura dowodowa?

Pomijając ograniczenia wynikające z odwołania do procedury dowodowej w powyższej definicji, taka procedura nie będzie jednak ani efektywna obliczeniowo, ani rozstrzygalna, ani nawet półrozstrzygalna, jak procedury dowodowe dla rachunku predykatów.

W przesłance powyższej reguły wnioskowania mamy bowiem stwierdzenie, że pewnej formuły nie da się udowodnić. To po pierwsze może być w ogóle niemożliwe do stwierdzenia. Zaś aby znaleźć pozytywne potwierdzenie tego faktu będzie na pewno konieczne przeprowadzenie globalnego wnioskowania na całej bazie danych, bo inaczej trudno byłoby stwierdzić, że czegoś nie da się udowodnić.

Dla odróżnienia, dowody w rachunku predykatów pierwszego rzędu mają charakter lokalny. Jeśli np. szczęśliwie wybierzemy od razu właściwe przesłanki to możemy uzyskać dowód w kilku krokach, nawet jeśli baza danych liczy tysiące faktów.

Problemy z metodami opartymi na logice

Podejście logiczne do reprezentacji wiedzy i rozwiązywania problemów budziło swojego czasu wiele emocji i nadziei na budowę wszechstronnych systemów sztucznej inteligencji. Istnieją jednak poważne przeszkody ograniczające zastosowanie tej metody do rozwiązywania problemów praktycznych:

- **eksplozja kombinatoryczna** procedury dowodowej — istnieją strategie usprawniające, jednak niewiele pomagają; jednocześnie trudno jest połączyć metody formalne z dostępną informacją heurystyczną
- **nierozstrzygalność** i gödlowska **niezupełność** rachunku predykatów
- wnioskowanie z uwzględnieniem **zmian** — rachunek sytuacji, logiki czasowe
  - pojawia się tu **problem tła** (*frame problem*) — poza określeniem co się zmieniło, konieczne jest śledzenie tego co się nie zmieniło
- wnioskowanie z użyciem informacji **niepełnej** i/lub **niepewnej** — inne wyzwanie dla metod formalnych, jednak nieodzowne w działaniu człowieka
  - uwzględnienie informacji niepełnej prowadzi do wnioskowania **niemonotonicznego**, którym ludzie posługują się sprawnie, podczas gdy tradycyjna logika matematyczna jest ściśle monotoniczna

Zastosowanie metod opartych na logice

Wymienione problemy z metodami opartymi na logice istotnie utrudniają ich wykorzystanie jako platformy budowy inteligentnych agentów. Powszechnie wykorzystywany w sztucznej inteligencji jest jedynie sam język logiki pierwszego rzędu jako język zapisu faktów.

Jednak w pewnych konkretnych zastosowaniach, w ograniczonych dziedzinach, powyższe problemy mają mniejsze znaczenie, i można skutecznie korzystać z tej metodologii.
Do tych zastosowań należą:

- synteza i weryfikacja programów, inżynieria oprogramowania,
- projektowanie i weryfikacja cyfrowej elektroniki obliczeniowej, w tym projektowanie układów VLSI,
- dowodzenie twierdzeń w matematyce; pozwala poszukiwać dowodów postulowanych twierdzeń, dla których nie udaje się znaleźć dowodu metodą tradycyjną.