Katedra Sztucznej Inteligencji i Matematyki Stosowanej
WI ZUT Szczecin

Sztuczna inteligencja

dr inż. Joanna Kolodziejczyk
jkolodziejczyk@zut.edu.pl
pokój nr 18 WI1
konsultacje: wtorki 10:00 - 12:00

21 maja 2022

Treść wykładu

Reprezentowanie wiedzy

Logika do reprezentacji wiedzy

Rachunek zdań

Rachunek predykatów pierwszego rzędu
- Składnia
- Wyrażanie wiedzy w Rachunek predykatów pierwszego rzędu
- Wnioskowanie w Rachunek predykatów pierwszego rzędu

Prolog
- Przejście ze składni logiki do Prologu
- Prolog - tylko klauzule Horna
- Ważne informacje o Prologu

Kodowanie wiedzy

Kodowanie wiedzy oznacza formalny sposób zapisu wiedzy (w postaci symboli), która ma być zgromadzona w systemie.

Do form reprezentacji można zaliczyć:

- postać regułowa (IF - THEN)
- drzewa decyzyjne
- tablice decyzyjne
- sieci semantyczne

Baza wiedzy

Kodowanie wiedzy powinno być wykonane w takiej formie, by możliwe było zbudowanie bazy wiedzy.

W wyniku kodowania powstanie baza wiedzy wspomagająca uczenie się i podejmowanie decyzji.

Główne wykorzystanie baz wiedzy:

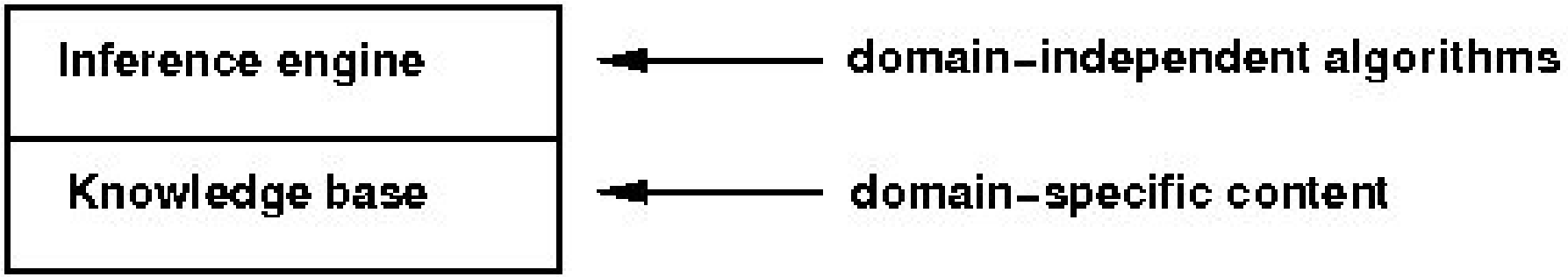
- diagnostyka
- uczenie się/ instruktaż
- interpretacja
- predykcja
- planowanie.

Wiedza a systemy jej przetwarzania

Człowiek ma wiedzę i na jej postawie wykonuje rozumowanie. W SI zawarcie wiedzy w systemie może zapewnić lepsze jego zachowanie.

System z wiedzą będzie dobrym rozwiązaniem do nie w pełni obserwowalnego środowiska (nie od razu muszą być znane wszystkie wejścia do systemu). Może dokonywać uogólnień doświadczeń z przestrzeni obserwowalnej. np. diagnoza lekarska lub rozumienie języka naturalnego.

Wiedza — jak pokazać w systemie



Baza wiedzy (KB)

zbiór zdań w języku formalnym lub inaczej w języku reprezentacji wiedzy.

Programowanie deklaratywne

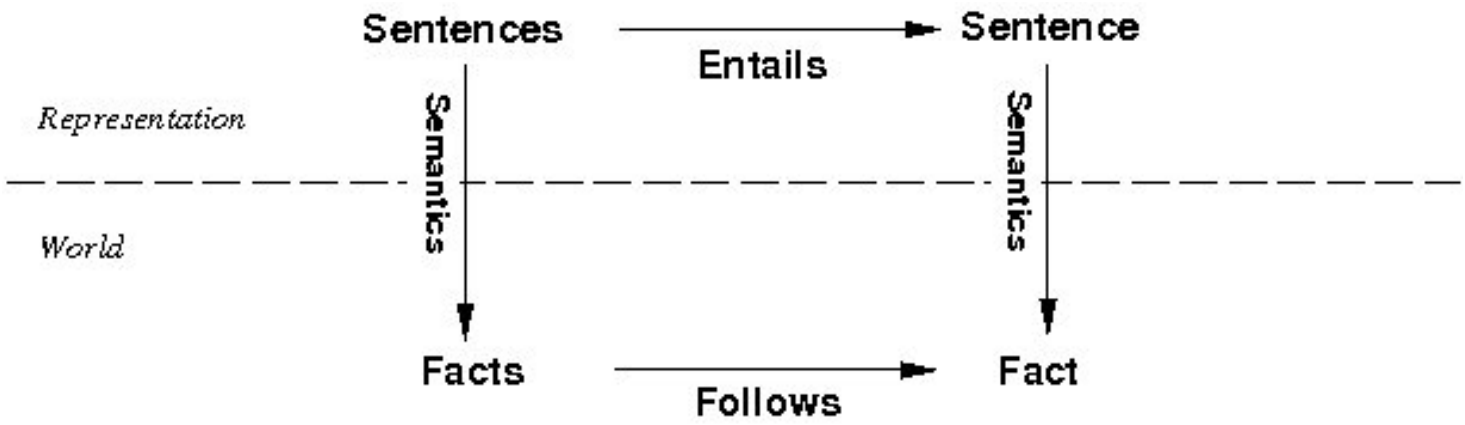
Podejście deklaratywne

podejście ma zapewnić wykonywanie następujących zadań:

- TELL — możliwość poinformowania systemu o nowej wiedzy (wprowadzanie nowych zdań).
- ASK — odpytywanie systemu co jest mu wiadome (odpowiedź powinna wynikać z bazy wiedzy).

ASK i TELL mogą wymagać wnioskowania czyli wyprowadzania nowych zdań na podstawie wcześniej zadanych.

Zależność pomiędzy reprezentacją i rzeczywistością



Representation — formalna reprezentacja wiedzy
World — rzeczywistość
Semnatics — znaczenie
Sentences — zdania reprezentujące wiedzę w jakimś języku formalnym
Facts — rzeczywiste zdarzenia, fakty
Follows — następstwo
Entails — konsekwencja

Logika — podstawy

Logika

Jest formalnym językiem reprezentowania wiedzy o obiektach, z pomocą której można wyciągać wnioski (konkluzje) o właściwościach tych obiektów.

Logika elementy:

- Składnia określa budowę zdań w danym języku formalny. Wnioskowanie musi uwzględniać manipulowanie i generowanie symboli zdań w określonej składni.
- Semantyka określa znaczenie wyrażenia. W logice semantyka definiuje prawdziwość (TRUE) każdego zdania w odniesieniu do rozpatrywanej rzeczywistości.
- system wnioskowania

Dwoistość logiki

- Jako system wnioskowania (*proof system*): Dany jest zbiór faktów (aksjomatów) i zbiór reguł wnioskowania. Celem jest ustalenie, które fakty wynikają z aksjomatów i reguł wnioskowania. W takim spojrzeniu na logikę wykonuje się czysto mechaniczne operacje na symbolach i nie patrzy się na znaczenie zdań, którymi się manipuluje. Nie oznacza to, że dowód nie wymaga kreatywności, ale znaczenie zdania jest w takim wypadku nieistotne.
- Jako teoria modeli (*model theory*): Zdania uzyskują znaczenie, co nazywa się interpretacją. W tym wypadku język logiki jest używany do sformalizowania właściwości struktur i określenia, kiedy zdanie jest prawdziwe. Teoria modeli zmusza do precyzyjnego definiowania pojęcia prawdy.

Poprawność systemu wnioskowania

Cechy systemu wnioskowania

- Niepodważalność (*sound*): każda dowiedziona w nim formuła jest ważna i poprawna.
- Zupełność (*complete*): każda ważna i poprawna formuła może być w nim dowiedziona.

Systemy zupełne: rachunek zdań, logika predykatów pierwszego rzędu.
Systemy niezupełne: logika predykatów drugiego rzędu.

Przykład składni i semantyki

- Arytmetyka to język formalny do reprezentowania zależności i operacji na liczbach.
- Składnia:
  $x + 2 \geq y$???
  $x2 + y >$???
- Semantyka
  $x + 2 \geq y$ jest prawdą witw,???
  $x + 2 \geq y$ jest prawdziwe w takiej rzeczywistości, gdzie ???
  $x + 2 \geq y$ jest fałszywe w takiej rzeczywistości, gdzie ???

Przykład składni i semantyki

- Arytmetyka to język formalny do reprezentowania zależności i operacji na liczbach.
- Składnia:
  $x + 2 \geq y$??? jest poprawnym wyrażeniem
  $x2 + y >$??? nie jest poprawnym wyrażeniem
- Semantyka
  $x + 2 \geq y$ jest prawdą witw,??? gdy liczba $x + 2$ jest nie mniejsza niż liczba y
  $x + 2 \geq y$ jest prawdziwe w takiej rzeczywistości, gdzie ???
  $x = 7$ i $y = 1$
  $x + 2 \geq y$ jest fałszywe w takiej rzeczywistości, gdzie ???$x = 0$ i $y = 6$

Logiki

Język	Co opisuje z rzeczywistości	Jakie może być przekonanie o faktach
Rachunek zdań	fakty	PRAWDA/ Fałsz/ nieznany
Logika predykatów pierwszego rzędu	fakty , obiekty, relacje	PRAWDA/ Fałsz/ nieznany
Logika temporalna	fakty , obiekty, relacje, czas	PRAWDA/ Fałsz/ nieznany
Teoria prawdopodobieństwa	fakty	przekonanie o faktach w skali 0...1
Logika rozmyta	stopień prawdziwości	przekonanie w skali 0...1

Składnia rachunku zdań

Rachunek zdań jest najprostszym składniowo systemem logicznym. Pewne założenia przenoszą się jednak na rachunek predykatów pierwszego rzędu.

Alfabet rachunku zdań

1. stałe: *True* i *False*
2. symbole oznaczające zdania (formuły, atomy): P, $Q1$
3. nawiasy okrągłe wokół zdania: $(P \wedge Q)$
4. zdania złożone przez kombinację symboli, stałych z pięcioma symbolami operacji

Rachunek zdań — składnia

1. Negacja $\neg S$ Jeżeli S (positive literal) jest formułą, to $\neg S$ (negative literal) jest formułą.
2. Koniunkcja $\wedge$ Jeżeli S_1 i S_2 to formuły, to $S_1 \wedge S_2$ jest formułą.
3. Dysjunkcja $\vee$ Jeżeli S_1 i S_2 to formuły, to $S_1 \vee S_2$ jest formułą.
4. Implikacja (warunek) $\Rightarrow$ Jeżeli S_1 i S_2 to formuły, to $S_1 \Rightarrow S_2$ jest formułą. Implikacja znana jest też jako reguła czyli zdania typu IF-THEN
5. Równoważność $\Leftrightarrow$ Jeżeli S_1 i S_2 to formuły, to $S_1 \Leftrightarrow S_2$ jest formułą. Czytane: „wtedy i tylko wtedy”.

Semantyka rachunku zdań

- Specyfikuje interpretację każdego symbolu i stałych i określa znaczenie zależności logicznych.
- Znaczenie symboli (ich interpretacja) jest dowolna. np. *P* może znaczyć „Paryż jest stolicą Francji”, czy też „Piotr ma niebieskie oczy”
- *P* może stać się *True*, jeżeli fakt, o którym mówi zaistniał.
- Zdania złożone mają takie znacznie, które wynika z ich składowych. Można zdania złożone traktować jak funkcje. Gdy podane są wartości wejściowe, to można obliczyć wartość wynikową.

Tablica prawdy

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	true	*false*	*false*	true	true
false	true	true	*false*	true	true	*false*
true	*false*	*false*	*false*	true	*false*	*false*
true	true	*false*	true	true	true	true

Interpretacja implikacji

Przykład

Ojciec obiecuje Jasiowi:

Jeśli $\overbrace{\text{jutro będzie ładna pogoda}}^{p}$, to $\overbrace{\text{pójdziemy na grzyby}}^{q}$.

- Obietnica jest implikacją $p \Rightarrow q$.
- Ojciec nie dotrzyma słowa tylko w jednym przypadku: jeżeli jutro będzie ładna pogoda (tzn. $p = 1$), a nie pójdą z Jasiem na grzyby (tzn. $q = 0$).
- Dlatego przyjmujemy, że implikacja $p \Rightarrow q$ jest fałszywa tylko wtedy, gdy $p = 1$ i $q = 0$.

Walidacja przez tablicę prawdy

Udowodnić zdanie: $((P \vee H) \wedge \neg H) \Rightarrow P$

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
false	*false*	*false*	*false*	true
false	true	true	*false*	true
true	*false*	true	true	true
true	true	true	*false*	true

Tautologie

$(\alpha \wedge \beta)$	$\Leftrightarrow$	$(\beta \wedge \alpha)$	*przemiennosc* $\wedge$
$(\alpha \vee \beta)$	$\Leftrightarrow$	$(\beta \vee \alpha)$	*przemiennosc* $\vee$
$((\alpha \wedge \beta) \wedge \gamma)$	$\Leftrightarrow$	$(\alpha \wedge (\beta \wedge \gamma))$	*lacznosc* $\wedge$
$((\alpha \vee \beta) \vee \gamma)$	$\Leftrightarrow$	$(\alpha \vee (\beta \vee \gamma))$	*lacznosc* $\vee$
$\neg(\neg\alpha)$	$\Leftrightarrow$	α	*eliminacja podwojnej negacji*
$(\alpha \Rightarrow \beta)$	$\Leftrightarrow$	$(\neg\beta \Rightarrow \neg\alpha)$	
$(\alpha \Rightarrow \beta)$	$\Leftrightarrow$	$(\neg\alpha \vee \beta)$	*eliminacja implikacji*
$(\alpha \Leftrightarrow \beta)$	$\Leftrightarrow$	$((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	*eliminacja rownowaznosci*
$\neg(\alpha \wedge \beta)$	$\Leftrightarrow$	$(\neg\alpha \vee \neg\beta)$	*De Morgan*
$\neg(\alpha \vee \beta)$	$\Leftrightarrow$	$(\neg\alpha \wedge \neg\beta)$	*De Morgan*
$(\alpha \wedge (\beta \vee \gamma))$	$\Leftrightarrow$	$((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	*rozdzielnosc* $\wedge$ *wzgledem* $\vee$
$(\alpha \vee (\beta \wedge \gamma))$	$\Leftrightarrow$	$((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	*rozdzielnosc* $\vee$ *wzgledem* $\wedge$

Schematy wnioskowania

Modus ponendo ponens (Modus Ponens)

Zapisujemy: $\dfrac{\begin{array}{c} P \Rightarrow Q \\ P \end{array}}{Q}$ lub jako tautologię: $(P \wedge (P \Rightarrow Q)) \Rightarrow Q)$

- Nazywana regułą dedukcji.
- Jeżeli prawdziwa jest reguła/implikacja (IF THEN) i część przesłankowa reguły P, to możemy wnioskować Q, czyli konsekwencję reguły.
- W sztucznej inteligencji nazywany wnioskowaniem w przód.
- np. $\dfrac{\begin{array}{c} \text{Jeżeli dziś jest niedziela, to jutro jest poniedziałek.} \\ \text{Dziś jest niedziela.} \end{array}}{\text{Jutro jest poniedziałek.}}$

Schematy wnioskowania

Modus ponendo tollens

Zapisujemy:
$$\frac{\neg(P \wedge Q) \quad P}{\neg Q}$$

lub jako tautologię: $(P \wedge \neg(P \wedge Q)) \Rightarrow \neg Q)$

- Czytamy jako: Albo ... albo
- np.
$$\frac{\text{Albo pójdę do kina, albo obejrzę telewizję.} \quad \text{Pójdę do kina.}}{\text{Nie obejrzę telewizji.}}$$

Schematy wnioskowania

Modus tollendo tollens

Zapisujemy:
$$\frac{\begin{array}{c} P \Rightarrow Q \\ \neg Q \end{array}}{\neg P}$$

- Nazywany zaprzeczeniem konsekwencji.
- Nie jest możliwe, by przesłanka była prawdziwa i konsekwencja była fałszywa.
- np.
$$\frac{\begin{array}{c} \text{Jeżeli pies wyczuje obcego, będzie warczał} \\ \text{Pies nie warczał.} \end{array}}{\text{Zatem pies nie wyczuł nikogo obcego.}}$$

Schematy wnioskowania

Modus tollendo ponens (sylogizm dysjunkcyjny/elimienacja dysjunkcji)

Zapisujemy:
$$\frac{P \vee Q \quad \neg P}{Q}$$

- Wiemy, że przynajmniej jedno ze stwierdzeń jest prawdziwe i wiemy, że inne nie jest prawdziwe, zatem wnioskujemy o prawdziwości drugiego.
- np.
$$\frac{\text{Pójdę do kina lub obejrzę telewizję} \quad \text{Nie pójdę do kina.}}{\text{Obejrzę telewizję}}$$

Schematy wnioskowania

Sylogizm warunkowy

Zapisujemy:
$$\frac{\begin{array}{c} P \Rightarrow Q \\ Q \Rightarrow R \end{array}}{P \Rightarrow R}$$

- np.
$$\frac{\begin{array}{c} \text{Jeżeli nie wstanę, to nie pójdę do pracy.} \\ \text{Jeżeli nie pójdę do pracy, nie zarobię.} \end{array}}{\text{Jeżeli nie wstanę, to nie zarobię.}}$$

- Przykład prowadzący do absurdalnych wniosków
$$\frac{\begin{array}{c} \text{Jeżeli Cezar pozostanie w domu, to nie zostanie zabity.} \\ \text{Jeżeli Cezar nie zostanie zabity, to wygłosi przemówienie w senacie.} \end{array}}{\text{Jeżeli Cezar pozostanie w domu, to wygłosi przemówienie w senacie.}}$$

- Błąd wnioskowania wynika z tego, że nie bierze się pod uwagę kontekstowego połączenia przesłanek i konkluzji.

Schematy wnioskowania

Eliminacja koniunkcji

Zapisujemy: $\dfrac{A \wedge B}{B}$ lub $\dfrac{A \wedge B}{A}$

- z iloczynu można wnioskować każdy czynnik
- np. $\dfrac{\text{Bob lubi jabłka i pomarańcze.}}{\text{Bob lubi jabłka.}}$

Schematy wnioskowania

Rezolucja

Zapisujemy: $\dfrac{A \vee B \quad \neg A \vee C}{B \vee C}$ z MP $\dfrac{A \Rightarrow B \quad A}{B}$ otrzymujemy $\dfrac{\neg A \vee B \quad A}{B}$

- Jeżeli A jest prawdziwe, to aby prawdziwe była druga przesłanka, to C musi być prawdziwe. A jeżeli A jest fałszywe, to aby pierwsza przesłanka była prawdziwa, B musi być prawdziwe.
- np. $\dfrac{\text{Jeżeli ktoś jest Grekiem to jest europejczykiem.} \quad \text{Homer jest Grekiem.}}{\text{Homer jest europejczykiem}}$

Koniunkcyjna postać normalna CNF

- Regułą rezolucji stosuje się tylko dla dysjunkcji literałów (zdań). Zatem konieczne będzie posiadanie bazy wiedzy w postaci takich właśnie sum logicznych.
- Każde zdanie w rachunku zdań jest logicznym równoważnikiem koniunkcji dysjunkcji literałów, to znaczy, że dowolne zdanie można na taką formę koniunkcyjną przekształcić.

Koniunkcyjna forma/postać normalna (Conjunctive Normal Form lub CNF)

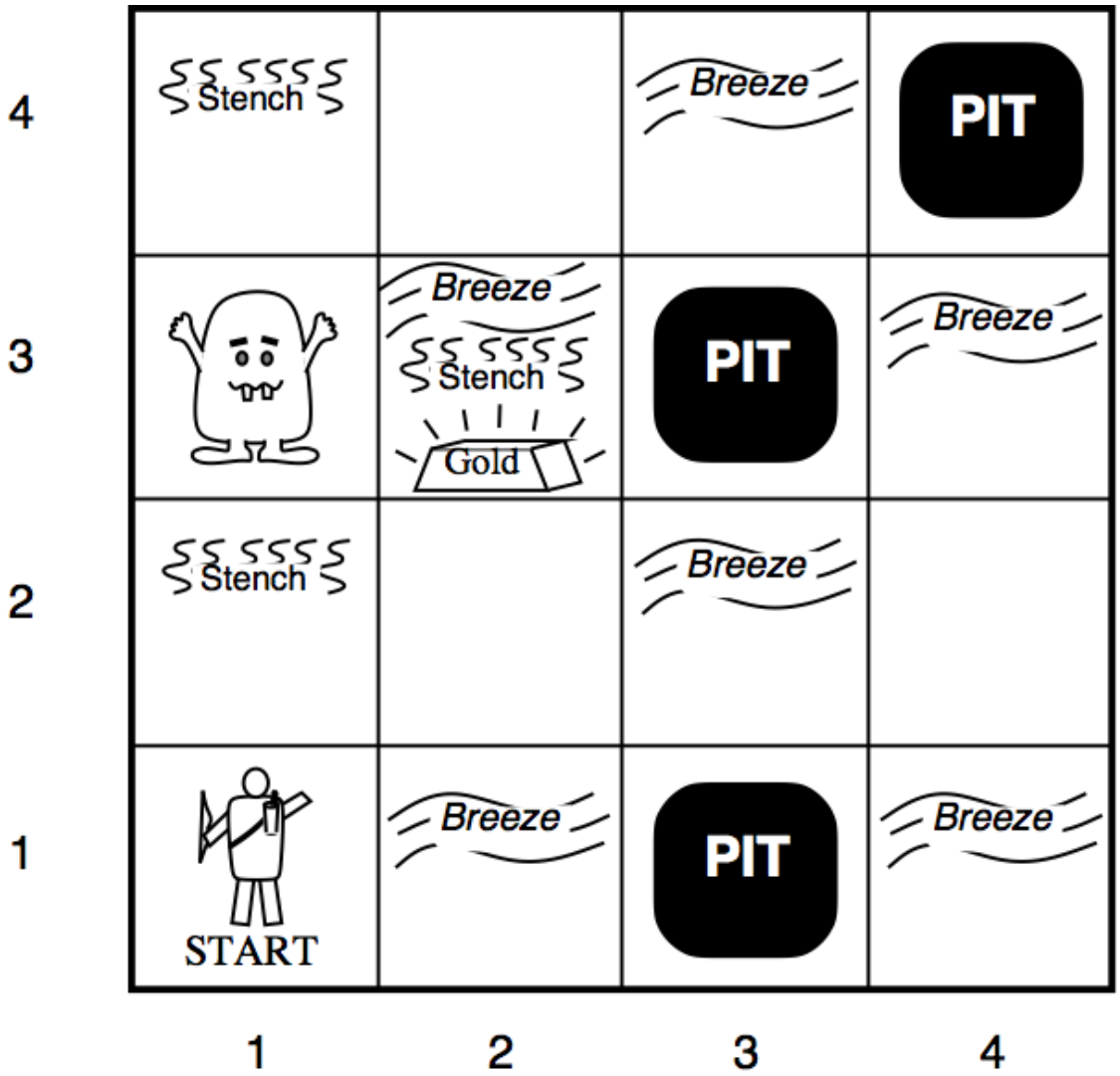
Jest to koniunkcja dysjunkcji literałów czyli iloczyn sum logicznych

np., $\overbrace{(A \vee \neg B)}^{klauzula1} \wedge \overbrace{(B \vee \neg C \vee \neg D)}^{klauzula2}$

Klauzula

Jest to suma literałów. Powyższy przykład zawiera dwie klauzule.

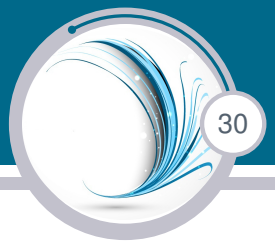
Wumpus by Michael Genesereth



Wumpus - opis

Miara osiągów	złoto +1000; śmierć -1000; -1 za każdy ruch; -10 za użycie strzały
Środowisko	Tablica 4×4 pomieszczeń. Agent zaczyna w polu $[1, 1]$ twarzą skierowaną w prawo. Położenie wumpusa i złota wybierane jest losowo z pominięciem pola startowego. Każde z pól z prawdopodobieństwem 0.2 może być dołkiem
Aktualizatory	Skręć w lewo, skręć w prawo, idź, chwytaj, upuść, strzelaj (do końca wiersza lub kolumny, lub wumpusa), umiera
Czujniki	Pola przylegające do wumpusa cuchną (*stench*). W polach przylegających do dołka czuć powiew (*breeze*). Blask jest w polu gdzie znajduje się złoto (*glitter*). Jeżeli uderzy się w ścianę to jest to sygnalizowane. Zabicie wumpusa jest oznajmiane jego krzykiem. [*Stench*, *Breez*, *None*, *None*, *None*]

Wumpus World w rachunku zdań



Plansza 4x4.

- Zasady "fizyka"gry:
  - $(B_{x,y} \Rightarrow (P_{x-1,y} \vee P_{x+1,y} \vee P_{x,y-1} \vee P_{x,y+1})$
  - $(S_{x,y} \Rightarrow (W_{x-1,y} \vee W_{x+1,y} \vee W_{x,y-1} \vee W_{x,y+1})$
- Przynajmniej jeden Wumpus na planszy

$$W_{1,1} \vee W_{1,2} \vee W_{1,3} \vee \cdots \vee W_{4,4}$$

- Co najwyżej jeden Wumpus na planszy (dla każdych dwóch sąsiednich pól) $\frac{n(n-1)}{2}$ reguł

$$\neg W_{1,1} \vee \neg W_{1,2}$$

- Nie można umrzeć na starcie

$$\neg P_{1,1}$$

$$\neg W_{1,1}$$

W sumie 155 zdań zawierających 64 symbole.

Algorytm konwersji do koniunkcyjnej postaci normalnej

Zdanie do przekształcenia: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Eliminacja $\Leftrightarrow$, zamień $A \Leftrightarrow B$ na $(A \Rightarrow B) \wedge (B \Rightarrow A)$.

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$$

2. Eliminacja $\Rightarrow$, zamień $A \Rightarrow B$ na $\neg A \vee B$.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

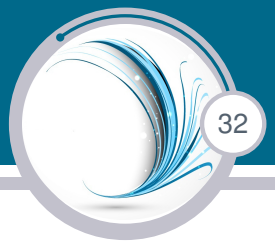
3. Przesunięcie $\neg$ do nawiasów stosując prawa de Morgana i podwójną negacje:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$$

4. Zastosowanie prawa rozdzielności $\vee$ względem $\wedge$:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

Algorytm rezolucji



Algorytm rezolucji

Nazywane dowodzenia przez sprzeczność (reducio ad absurdum). Aby wykazać iż A jest spełnione (wynika) z bazy wiedzy KB wykazane zostanie, że $KB \wedge \neg A$ jest niespełnialne, czyli prowadzi do zadnia pustego. Uzyskanie sprzeczności potwierdza postawione założenie.

- Baza wiedzy musi zostać uprzednio przekształcona do postaci CNF.
- Jeżeli dowodzi się złożone zdanie logiczne, to też musi zostać przekształcone na postać CNF.

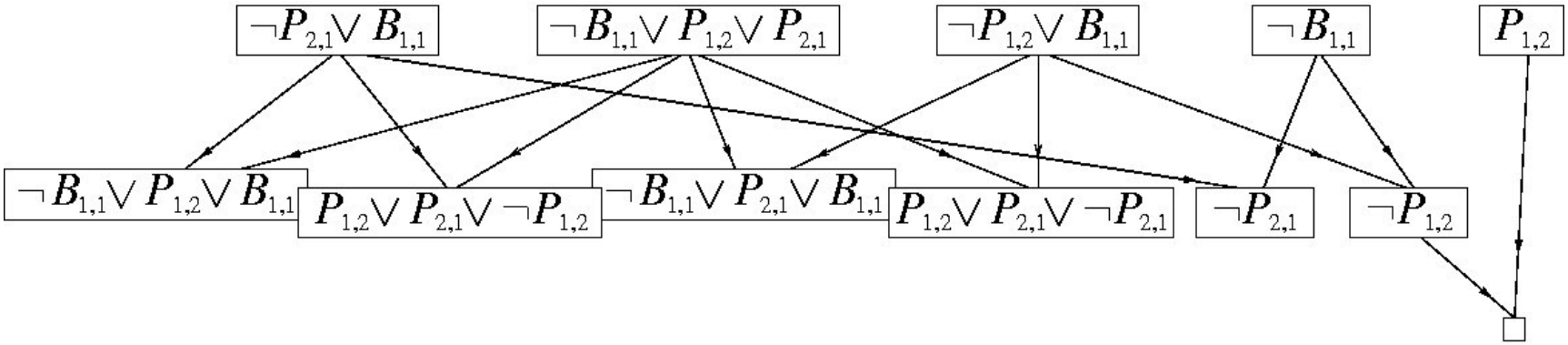
Algorytm rezolucji

1. Zamień wszystkie zdania w bazie wiedzy i zdania do udowodnienia na postać CNF i utwórz z nich jeden zbiór klauzul roboczych.
2. Zastosuj regułę rezolucji do wszystkich możliwych par klauzul, które zawierają literały komplementarne. W wyniku zastosowania reguły rezolucji powstaną resolwenty (klauzule bez literałów komplementarnych).
3. Jeżeli resolwenty nie istnieją w zbiorze klauzul roboczych dodaj je do niego.
4. Idź do kroku 2 lub zakończ, gdy uzyskasz zdanie puste (udowodniono hipotezę) lub nie tworzą się żadne nowe klauzule (nie można dowieźć hipotezy z bazy wiedzy).

Przykład dowodzenia z użyciem rezolucji

$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$ Dowodzimy: $\alpha = \neg P_{1,2}$

Po konwersji $KB \wedge \neg\alpha$ na CNF otrzymujemy pierwszy wiersz na schemacie.



Drugi wiersz powstaje jako resolwenty z połączeń wszystkich klauzul z pierwszego wiersza. Ostatecznie dwie klauzule zostają połączone prowadząc do klauzuli pustej (sprzeczności) oznaczonej jako mały kwadrat. Zatem dowiedliśmy, że α.

Rachunek zdań: wady i zalety

- Rachunek zdań jest deklaratywny.
- Rachunek zdań dopuszcza częściową/alternatywnę/zanegowaną informacje (w przeciwieństwie do większości struktur danych i baz danych).
- Rachunek zdań jest zależny od składni: znaczenie $B_{1,1} \wedge P_{1,2}$ wynika ze znaczenia $B_{1,1}$ i $P_{1,2}$.
- Składnia w rachunku zdań jest niezależne od kontekstu (w przeciwieństwie do języka naturalnego).
- Rachunek zdań ma bardzo ograniczoną moc wyrażania (w przeciwieństwie do języka naturalnego), np. nie da się wyrazić zdania „pułapki powodują wiatr w sąsiednich polach” jako reguły. Trzeba wprost napisać oddzielny fakt.

Porównanie z rachunkiem zdań

Rachunek zdań

- ograniczona ekspresja: świat składa się tylko z faktów
- literały opisują fakty, bądź reguły o konkretnych obiektach
- treść rozpatrywanych zdań nie ma znaczenia, istotna jest jedynie ich wartość logiczna.

Rachunek predykatów pierwszego rzędu

- bardziej uniwersalna: można wyrazić wszystko, co się da zaprogramować
- pozwala reprezentować i wyrażać bardziej ogólne fakty i reguły
- zachowuje wszystkie poczynione w ramach klasycznego rachunku zdań ustalenia.

Składnia

Symbole stałe

nazywają pojedyncze rzeczy, jeden obiekt, są najczęściej argumentami predykatów. np. *Jan*, *A*, *Marcus*.

Symbole predykatów

Reprezentują relacje między obiektami np. $P(A)$, $R(A, B)$. Oznaczamy je wielkimi literami: „P;Q;R; S” lub wyrazami: *relacja*, *rodzic*’. *rodziec* jest symbolem predykatu binarnego, który zachowuje relację (lub nie) pomiędzy dwoma obiektami.

Symbole funkcji

Relacja funkcyjna jest zachowana dla dokładnie jednego obiektu w relacji np. *cosinus*(5), *ojciec*(*Jan*). Oznaczamy je wielkimi literami: „P;Q;R; S” lub wyrazami: *relacja*, *rodzic*’. *rodzic* jest symbolem predykatu binarnego, który zachowuje relację (lub nie) pomiędzy dwoma obiektami.

Składnia cd

Zdanie atomowe

Wyraża fakt odnoszący się do obiektów i relacji pomiędzy nimi opisanymi symbolami predykatów, np. brat(Ryszard, Jan), małżeństwo (ojciec(Ryszard), matka(Jan)). Zdanie atomowe ma wartość logiczną *TRUE*, jeżeli relacja opisana symbolem predykatu zachodzi pomiędzy obiektami podanymi jako argumenty relacji.

Operatory zdaniotwórcze

Takie same jak w rachunku zdań: $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$

Równość termów

$term_1 = term_2$ jest prawdziwe w danej interpretacji jeżeli $term_1$ i $term_2$ odnoszą się do tego samego obiektu, gdzie $term_i$ to symbol stałej lub symbol funkcji. $=$ wykorzystuje się w zdaniach złożonych.
ojciec(Jan) = Henryk

Składnia cd

Kwantyfikatory

Określają pewną liczbę indywiduów, którym przysługuje pewna własność lub które pozostają w relacji.

- Kwantyfikator ogólny (duży/uniwersalny) $\forall x$ „Dla każdego x”
- Kwantyfikator szczególny (mały/egzystencjonalny) $\exists y$ „Istnieje takie y”

Przykład na zdania z małym kwantyfikatorem

- Istnieją ludzie, którzy są dobrzy.
  $\exists x(\mathit{czlowiek}(x) \wedge \mathit{dobry}(x))$
- Istnieją ludzie, którzy nie są dobrzy.
  $\exists x(\mathit{czlowiek}(x) \wedge \neg \mathit{dobry}(x))$
- Nie tylko ludzie są dobrzy.
  $\exists x(\neg \mathit{czlowiek}(x) \wedge \mathit{dobry}(x))$

Mały kwantyfikator nie łączy się z implikacją!

Przykład na zdania z wielkim kwantyfikatorem

- Wszyscy ludzie są dobrzy
  $\forall x(czlowiek(x) \Rightarrow dobry(x))$
- Żaden człowiek nie jest dobry
  $\forall x(czlowiek(x) \Rightarrow \neg dobry(x))$
- Tylko ludzie są dobrzy.
  $\forall x(czlowiek(x) \Rightarrow dobry(x))$

Duży kwantyfikator nie łączy się z iloczynem logicznym!

Własności kwantyfikatorów

1. Równoważność zapisów $\forall x \forall y$ i $\forall y \forall x$ i $\forall x, y$
2. Równoważność zapisów $\exists x \exists y$ i $\exists y \exists x$
3. $\forall x \exists y$ nie jest równoważne $\exists x \forall y$
4. Prawa de Morgana dla kwantyfikatorów
   - $\forall x P(x) \Leftrightarrow \neg \exists x \neg P(x)$
   - $\exists x P(x) \Leftrightarrow \neg \forall x \neg P(x)$
   - $\neg \forall x P(x) \Leftrightarrow \exists x \neg P(x)$
   - $\forall x \neg P(x) \Leftrightarrow \neg \exists x P(x)$

Przykład bardziej złożony

1. Marcus był człowiekiem. $man(Marcus)$
2. Marcus był mieszkańcem Pompei. $pompeian(Marcus)$
3. Wszyscy mieszkańcy Pompei byli Rzymianami.
   $\forall x\ pompeian(x) \Rightarrow roman(x)$
4. Cezar był władcą. $ruler(Caesar)$
5. Wszyscy Rzymianie byli lojalni wobec Cezara, lub nienawidzili go. $\forall x\ roman(x) \Rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$
6. Każdy człowiek jest wobec kogoś lojalny. $\forall x \exists y\ loyalto(x, y)$
7. Ludzie starają się zgładzić tylko takiego władcę, wobec którego nie są lojalni.
   $\forall_{x,y}\ man(x) \wedge ruler(y) \wedge trytoassassinate(x, y) \Rightarrow \neg loyalto(x, y)$
8. Marcus próbował zgładzić Cezara.
   $trytoassassinate(Marcus, Caesar)$

Analiza przykładu

- Nie uwzględniono czasu przeszłego. W dalszych rozważaniach nie będzie to miało znaczenia. Wszystko dzieje się w przeszłości.
- Imiona nie zawsze wskazują na konkretne indywiduum. Dokładność wymaga sporej ilości wiedzy przechowywanej w bazie.
- Problem zasięgu. Czy dla każdego istnieje ktoś dla kogo jest się lojalnym? Czy są to różne osoby? Czy istnieje ktoś, wobec kogo wszyscy są lojalni?
- Zdanie 7 można też zapisać jako:
  $\forall x, y\ man(x) \wedge ruler(y) \wedge \neg loyalto(x, y) \Rightarrow trytoassassinate(x, y)$

Inżynieria wiedzy w rachunek predykatów pierwszego rzędu

1. Zidentyfikuj zadanie.
2. Zgromadź wymaganą wiedzę.
3. Wybierz słownik predykatów, funkcji i stałych.
4. Zakoduj ogólną wiedzę dotyczącą zadania.
5. Zakoduj specyficzną wiedzę dotyczącą zadania.
6. Postaw pytania do procedury wnioskowania i otrzymaj odpowiedzi.
7. Usuń błędy z bazy wiedzy.

Dowodzenie w logice pierwszego rzędu

Dla rachunku zdań przedstawione zostały reguły wnioskowania takie jak modus ponens, rezolucji czy eliminacja koninunkcji i te same reguły są poprawne dla logiki predykatów.

Wymaga się jednak dodatkowych operacji, które poradzą sobie z kwantyfikatorami. Jedną z nich jest UNIFIKACJA.

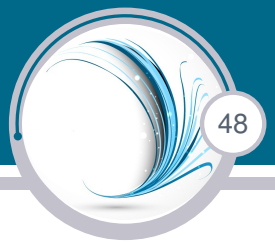
Unifikacja

Algorytm unifikacji jest rekurencyjną procedurą, porównującą dwa predykaty i odkrywającą, czy istnieje zbiór podstawień, które sprawią, że termy staną się identyczne.

Przykład na unifikację p i q. Wynikiem działania unifikacji jest θ.

p	q	θ
Knows(John,x)	Knows(John,Jane)	Jane/x
Knows(John,x)	Knows(y,OJ)	OJ/x,John/y
Knows(John,x)	Knows(y,Mother(y))	John/y,Mother(John)/x
Knows(John,x)	Knows(x,OJ)	fail

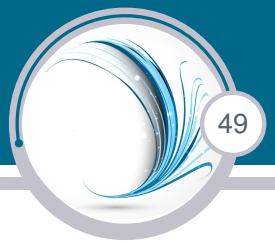
Unifikacja - MGU



MGU — Most General Unifier

- dane są fakty: $hate(x, y)$ i $hate(Marcus, z)$
- Unifikacja może zwrócić listę podstawień:
  1. (Marcus/x, z/y)
  2. (Marcus/x, y/z)
  3. (Marcus/x, Caesar/y, Caesar/z)
  4. (Marcus/x, Paulus/y, Paulus/z)
- Podstawienia 1 i 2 są bardziej ogólne niż 3 i 4. Ostatecznie do dalszej rezolucji poszukuje się jak najbardziej ogólnego unifikatora.

Reguła rezolucji w logice pierwszego rzędu



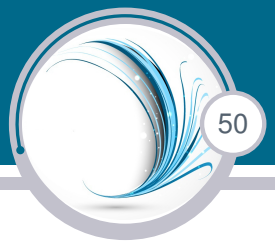
Reguła rezolucji

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

Gdzie: l_i i m_i to predykaty lub termy Wymagania:

- wymaga ze względu na argumenty relacji zastosowania unifikacji
- zdania muszą być podane w koniunkcyjnej postaci normalnej (CNF).

Reguła rezolucji w logice pierwszego rzędu



Przykład

$$\frac{(\neg rich(x) \vee happy(x)) \quad \wedge \quad (rich(Richard))}{happy(Richard)}$$

- gdzie w wyniku unifikacji: $\theta = Richard/x$
- zdania dane są w koniunkcyjnej postaci normalnej (CNF).

Konwersja do postaci normalnej koniunkcyjnej w logice pierwszego rzędu

1. Eliminacja $\Leftrightarrow$ zamiana na iloczyn implikacji.
2. Eliminacja $\Rightarrow$ zamiana na sumę.
3. Przesunięcie $\neg$ do nawiasów stosując prawa de Morgana i podwójną negacje.
4. Standaryzacja zmiennych np.: zamień $\forall x P(x) \vee \forall x Q(x)$ na $\forall x P(x) \vee \forall y Q(y)$.
5. Przesunięcie wszystkich kwantyfikatorów np.: zamień $\forall x P(x) \vee \forall y Q(y)$ na $\forall x \forall y P(x) \vee Q(y)$.
6. Usunięcie małego kwantyfikatora - skolemizacja: np. zamień $\forall x \exists y P(x, y)$ na $\forall x P(x, S(x))$.
7. Usunięcie wielkiego kwantyfikatora — po prostu dalej się je pomija.
8. Zastosowanie prawa rozdzielności $\vee$ względem $\wedge$.
9. Kolejna standaryzacja zmiennych — zróżnicowanie zmiennych w poszczególnych klauzulach.

Przykład konwersji

Każdy Rzymianin, który zna Marcusa nienawidzi Cezara, albo myśli, że każdy, kto nienawidzi kogokolwiek jest głupcem.

Przykład konwersji

Każdy Rzymianin, który zna Marcusa nienawidzi Cezara, albo myśli, że każdy, kto nienawidzi kogokolwiek jest głupcem.

$$\forall x(\mathit{roman}(x) \wedge \mathit{know}(x, \mathit{Marcus})) \Rightarrow$$

$$(\mathit{hate}(x, \mathit{Caesar}) \vee (\forall y(\exists z\ \mathit{hate}(y, z)) \Rightarrow \mathit{thinkcrazy}(x, y)))$$

Kolejne etapy konwersji

Etap 2 $\forall x \neg(roman(x) \wedge know(x, Marcus)) \vee (hate(x, Caesar) \vee (\forall y \neg(\exists z\ hate(y, z)) \vee thinkcrazy(x, y)))$

Etap 3 $\forall x(\neg roman(x) \vee \neg know(x, Marcus)) \vee (hate(x, Caesar) \vee (\forall y(\forall z \neg hate(y, z) \vee thinkcrazy(x, y)))$

Etap 5 $\forall x \forall y \forall z(\neg roman(x) \vee \neg know(x, Marcus)) \vee (hate(x, Caesar) \vee (\neg hate(y, z) \vee thinkcrazy(x, y)))$

Etap 7 $\neg roman(x) \vee \neg know(x, Marcus) \vee hate(x, Caesar) \vee \neg hate(y, z) \vee thinkcrazy(x, y)))$

Przykład — konwersja na CNF

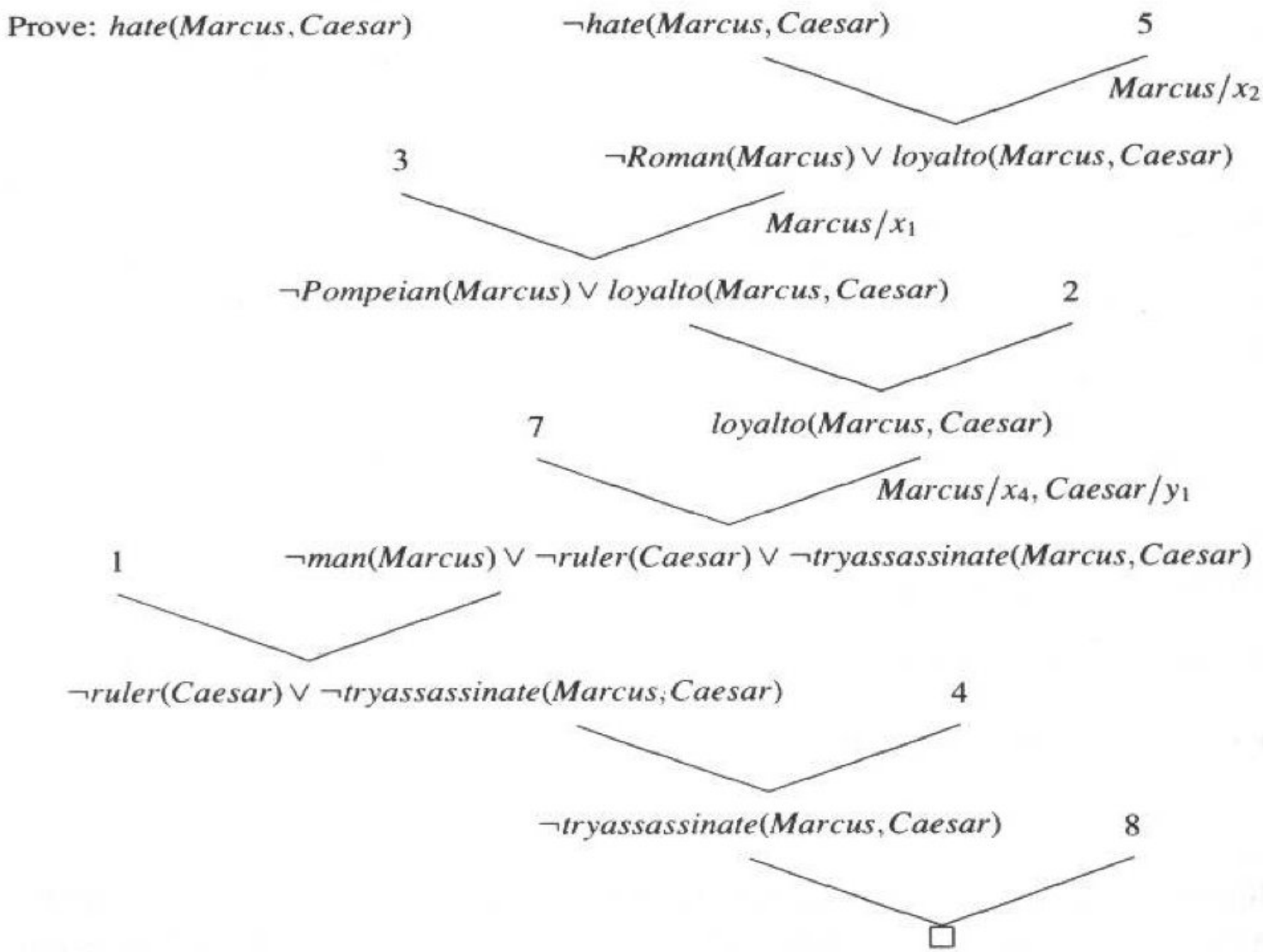
W wyniku konwersji bazy wiedzy ze slajdu 43 na postać CNF otrzymujemy:

1. $man(Marcus)$
2. $pompeian(Marcus)$
3. $\neg pompeian(x_1) \vee roman(x_1)$
4. $ruler(Caesar)$
5. $\neg roman(x_2) \vee loyalto(x_2, Caesar) \vee hate(x_2, Caesar)$
6. $loyalto(x_3, f(x_3))$
7. $\neg man(x_4) \vee \neg ruler(y_1) \vee \neg tryassassinate(x_4, y_1) \vee \neg loyalto(x_4, y_1)$
8. $tryassassinate(Marcus, Caesar)$

Algorytm rezolucji

1. Konwersja zdań z bazy wiedzy na klauzule (CNF).
2. Zaneguj P (predykat do udowodnienia) i przekształć na klauzulę. Dodaj do zbioru klauzul z bazy wiedzy.
3. Powtarzaj, do osiągnięcia klauzuli pustej, lub zatrzymaj, gdy nie ma postępu:
   - 3.1 Wybierz 2 klauzule (rodzicielskie).
   - 3.2 Porównaj je. Klauzula wynikowa - rezolwenta jest sumą logiczną wszystkich literałów obu klauzuli rodzicielskich z zastosowaniem odpowiednich podstawień z następującymi wyjątkami: Jeżeli istnieje para unifikowalnych literałów T_1 i $\neg T_2$ i T_1 i $\neg T_2$ to literały komplementarne. Zastosuj zbiór podstawień wynikający z unifikacji do produkcji rezolwenty. Jeżeli liczba literałów komplementarnych jest „> 1”, to w rezolwencie wybieramy tylko jedną z nich.
   - 3.3 Jeżeli rezolwenta jest klauzulą pustą, to dowód zkończono. Jeżeli nie, to dodaj klauzulę do zbioru klauzul.

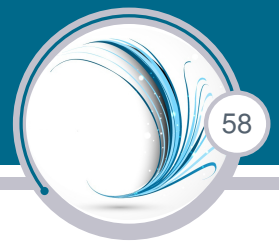
Przykład rezolucji



Co wiemy

1. Jaka jest składnia i semantyka rachunku predykatów pierwszego rzędu.
2. Jak działa algorytm rezolucji (Jeżeli teza $\{A_1, A_2, ..., A_n\}$ jest niesprzeczna, formuła B jest wnioskiem z $\{A_1, A_2, ..., A_n\}$ wtedy i tylko wtedy, gdy teza $\{A_1, A_2, ..., A_n, \neg B\}$ jest sprzeczna).
3. Czym jest postać klauzulowa.
4. Jak działa unifikacja.

Popatrzmy na klauzule inaczej



Zakładamy klauzule

$$(p_1(x) \vee p_2(x) \vee \cdots \vee p_n(x)) \vee (\neg q_1(x) \vee \neg q_2(x) \vee \cdots \vee \neg q_n(x))$$

Jest ona równoważna

$$(p_1(x) \vee p_2(x) \vee \cdots \vee p_n(x)) \vee (\neg(q_1(x) \wedge q_2(x) \wedge \cdots \wedge q_n(x))$$

Co jest z kolei równoważne

$$(p_1(x) \vee p_2(x) \vee \cdots \vee p_n(x)) \Leftarrow (q_1(x) \wedge q_2(x) \wedge \cdots \wedge q_n(x))$$

Jeżeli oznaczymy ($\wedge$ jako **,**) a ($\vee$ jako **;**) a ($\Leftarrow$ jako **:-**)

$$(p_1(x); p_2(x); \ldots; p_n(x)) :- (q_1(x), q_2(x), \ldots, q_n(x))$$

Reguły i fakty w nowej formule

- Reguła zawiera znak: **:-**. Po lewej stronie znaku znajduje się głowa, a po prawie treść reguły.
- Fakty są głowami klauzul bez treści, gdyż reprezentują aksjomaty.
- np.. kobieta(ala) w postaci klauzuli ma taką samą postać, co jest równoważne zapisowi: kobieta(ala) :- .

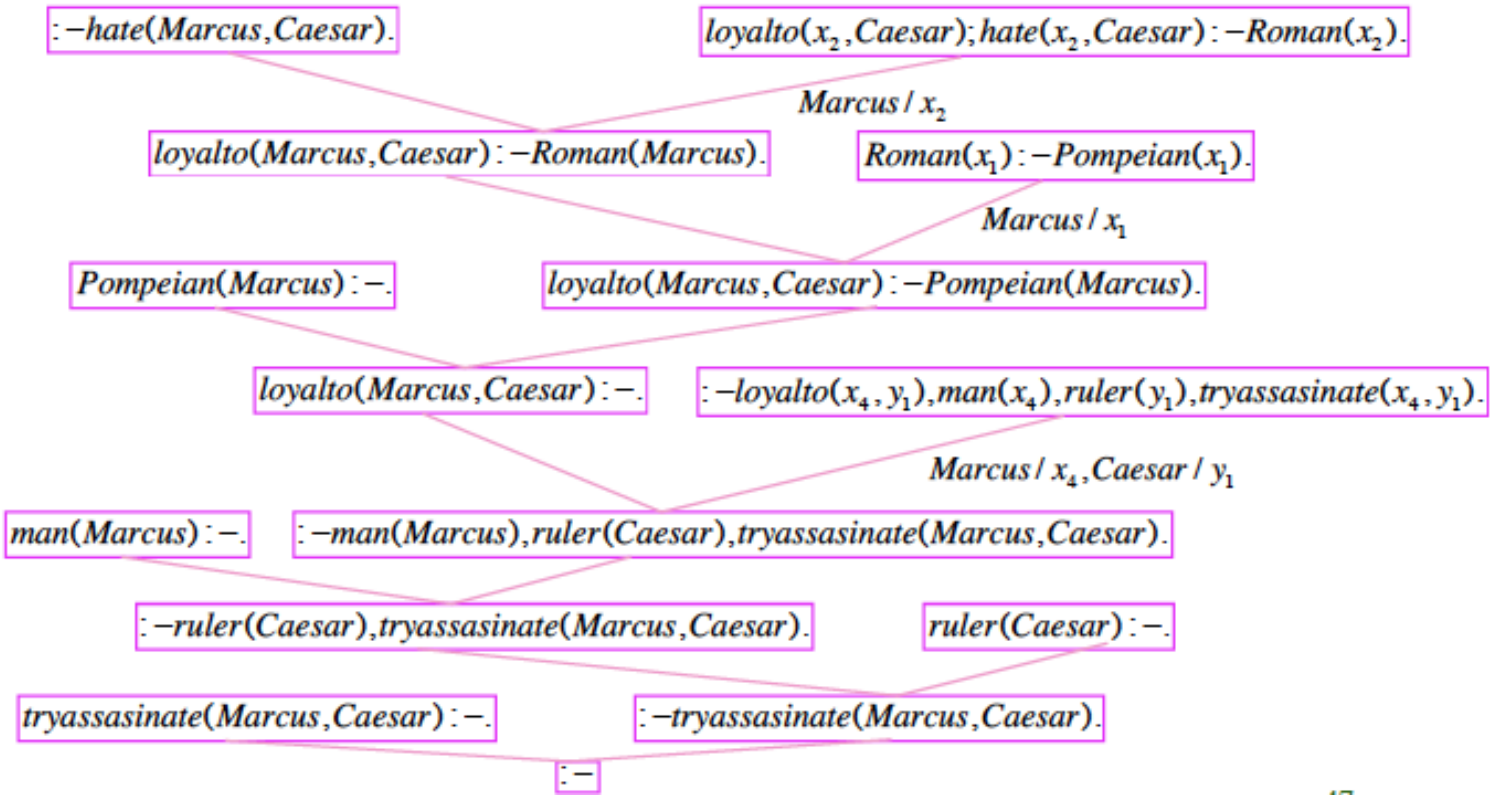
Przykład w nowej składni

W wyniku konwersji zdań CNF ze slajdu 54 otrzymujemy:

1. $man(Marcus) :-$
2. $pompeian(Marcus) :-$
3. $roman(x_1) :- pompeian(x_1)$
4. $ruler(Caesar) :-$
5. $loyalto(x_2, Caesar); hate(x_2, Caesar) :- roman(x_2)$
6. $loyalto(x_3, f(x_3)) :-$
7. $loyalto(x_4, y_1) :- man(x_4), ruler(y_1), tryassassinate(x_4, y_1)$
8. $tryassassinate(Marcus, Caesar) :-$

Rezolucja na nowej składni

Dowodzimy, że hate(Marcus, Caesar), zatem do bazy wprowadzamy zanegowaną hipotezę.



Klauzula Horna

Kluzula Horna

Jest to klauzula zawierająca co najwyżej jeden niezanegowany predykat.
Klauzule Horna dzieli się na:

1. z głową posiadające jeden niezanegowany predykat
2. bez głowy bez niezanegowanego predykatu

Klauzulą Horna nie jest:
$loyalto(x_2, Caesar); hate(x_2, Caesar) :- roman(x_2)$

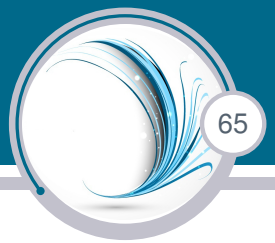
Zbiory klauzul Horna jako baza wiedzy

- Wszystkie klauzule poza jedną posiadają głowę.
- Baza wiedzy zawiera klauzule z głową.
- Cel wywodu (pytanie) jest klauzulą bez głowy.
- Istnienie jednej klauzuli bez głowy daje szansę uzyskania sprzeczności czyli uzyskanie pustej głowy i pustej treści.
- Istnienie wielu klauzul bez głowy jest zbędne, ponieważ każdy dowód można wyprowadzić przy użyciu co najwyżej jednej klauzuli bez głowy. Zatem klauzula pusta wynika tylko z klauzul z głową i jednej bez głowy.

Prolog - programowanie w logice

- Program zawiera informacje o danych z odpowiednią ich interpretacją. (Symboliczne opisanie wiedzy).
- Baza wiedzy zawiera oprócz danych (faktów) interpretowalne reguły (zależności pomiędzy danymi).
- Aby wyrazić reguły i fakty stosuje się logikę predykatów, a dokładnie klauzule Horna.
- Stosując zasady logiki tworzy się system zawierający fakty i reguły.
- Zadaje się pytania dotyczące zebranej wiedzy.

Przykład



logika	Prolog
$\forall x pet(x) \wedge small(x) \Rightarrow apartment(x)$	`apartment(X) :- pet(X),small(X).`
$\forall x cat(x) \vee dog(x) \Rightarrow pet(x)$	`pet(X):-cat(X); dog(X).`
$\forall x poodle(x) \Rightarrow dog(x) \wedge small(x)$	`small(X) :- poodle(X).` `dog(X) :- poodle(X).`
$poodle(fluffy)$	`poodle(flaffy).`

Cechy programowania w Prologu

- Źródło w Prologu to zbiór klauzul Horna z głową.
- Do wnioskowania stosuje się rezolucję, która dopasowuje cel (klauzulę bez głowy) do innych klauzul z głową, próbując wszystkie kombinacje klauzul z bazy.
- Nie wykorzystuje się powstałych pośrednio wniosków (nie dopisuje się ich ani tymczasowo, ani na stałe).
- Jeżeli cel jest złożony, np.. : $parent(X, Y), female(X).$, to Prolog dowodzi pierwszy z lewej, dopiero później po uzyskaniu dla niego pustej klauzuli przechodzi do kolejnego celu. Na końcu składa podcele.
- Algorytm sprawdzający wszystkie możliwości to strategia w głąb - DFS (Depth First Search).

Niuanse

Dlaczego $\forall x poodle(x) \Rightarrow dog(x) \wedge small(x)$ to w Prologu

```
dog(X) :- poodle(X).
small(X) :- poodle(X).
```

$$\forall x poodle(x) \Rightarrow dog(x) \wedge small(x) \Leftrightarrow$$
$$\forall x \neg poodle(x) \vee (dog(x) \wedge small(x)) \Leftrightarrow$$
$$\neg poodle(x) \vee (dog(x) \wedge small(x)) \Leftrightarrow$$
$$(\neg poodle(x) \vee dog(x)) \wedge (\neg poodle(x) \vee small(x)) \Leftrightarrow$$
$$dog(x) \vee \neg poodle(x) \Leftrightarrow dog(x) \Leftarrow poodle(x)$$
$$small(x) \vee \neg poodle(x) \Leftrightarrow small(x) \Leftarrow poodle(x)$$

```
dog(X) :- poodle(X).
small(X) :- poodle(X).
```

Unifikacja w PROLOGU

Proces kojarzenia zmiennych i wartości. Zmienna, której przypisano stałą wartość nazywa się ukonkretnioną.

Reguły unifikacji

- Stała może być zunifikowana tylko ze sobą.
- Dwie struktury(predykaty) mogą ze sobą zunifikować wtedy i tylko wtedy, gdy nazwy funktorów są takie same, mają taką samą liczbę argumentów, dopiero w kolejnym kroku rekurencyjnie unifikuje się ich argumenty.
- Zmienne unifikują się ze wszystkim.

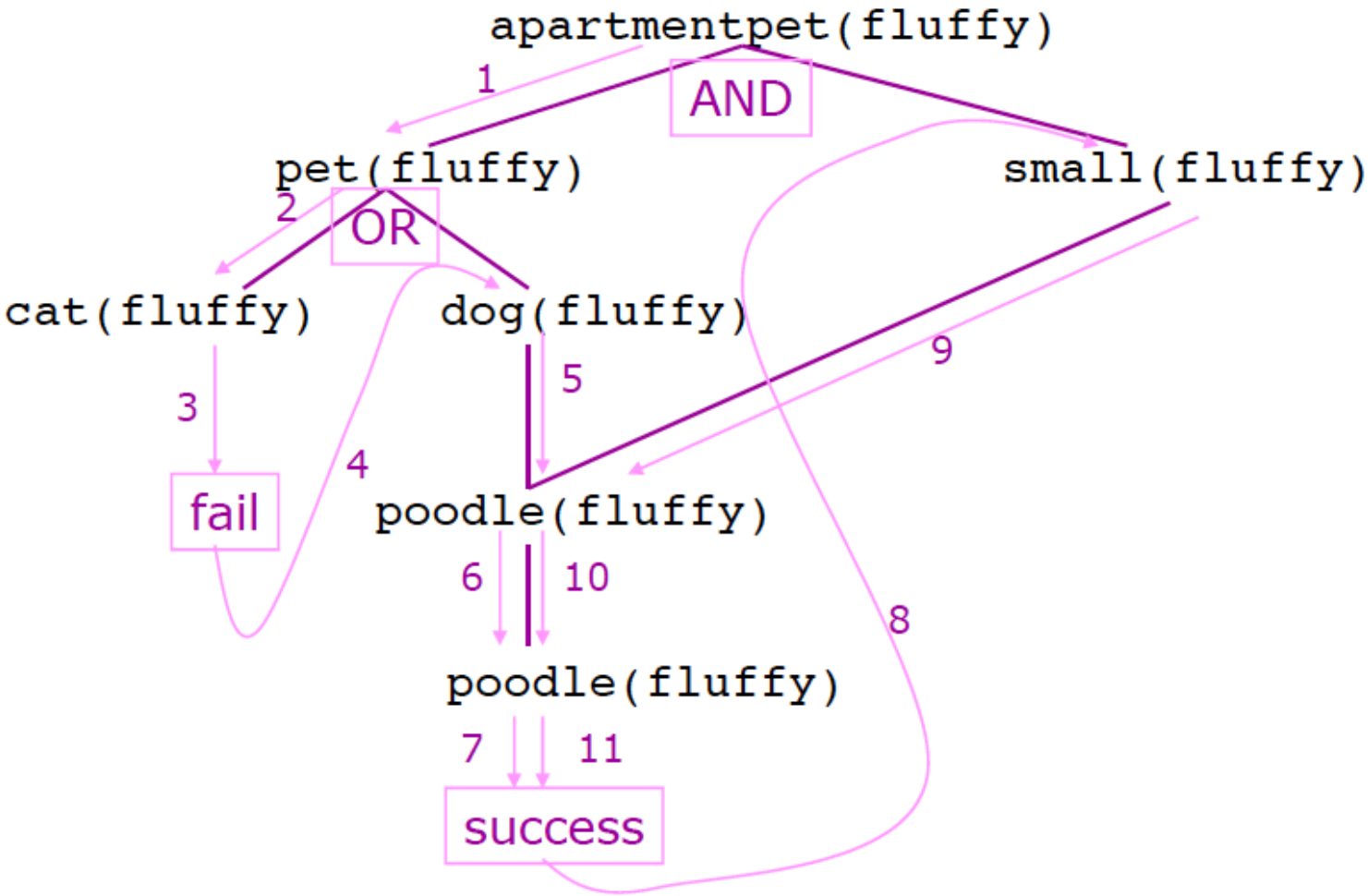
Kolejność wnioskowania

Dany jest plik źródłowy

```
apartmentpet(X) :- pet(X), small(X).
pet(X) :- cat(X).
pet(X) :- dog(X).
dog(X) :- poodle(X).
small(X) :- poodle(X).
poodle(fluffy).
```

Sprawdzamy, czy `apartmentpet(fluffy)` jest prawdziwe.

Przykład rezolucji



Kolejność kluzul

- Zachowanie interpretera jest przewidywalne, bo źródło jest przeszukiwane zgodnie z algorytmem DFS.
- Szczególnie istotnie wpływa to na predykaty rekurencyjne (głowa i treść reguły zawiera ten sam funktor).
- Niewłaściwa kolejność definicji klauzul może zaowocować nieskończoną pętlą.

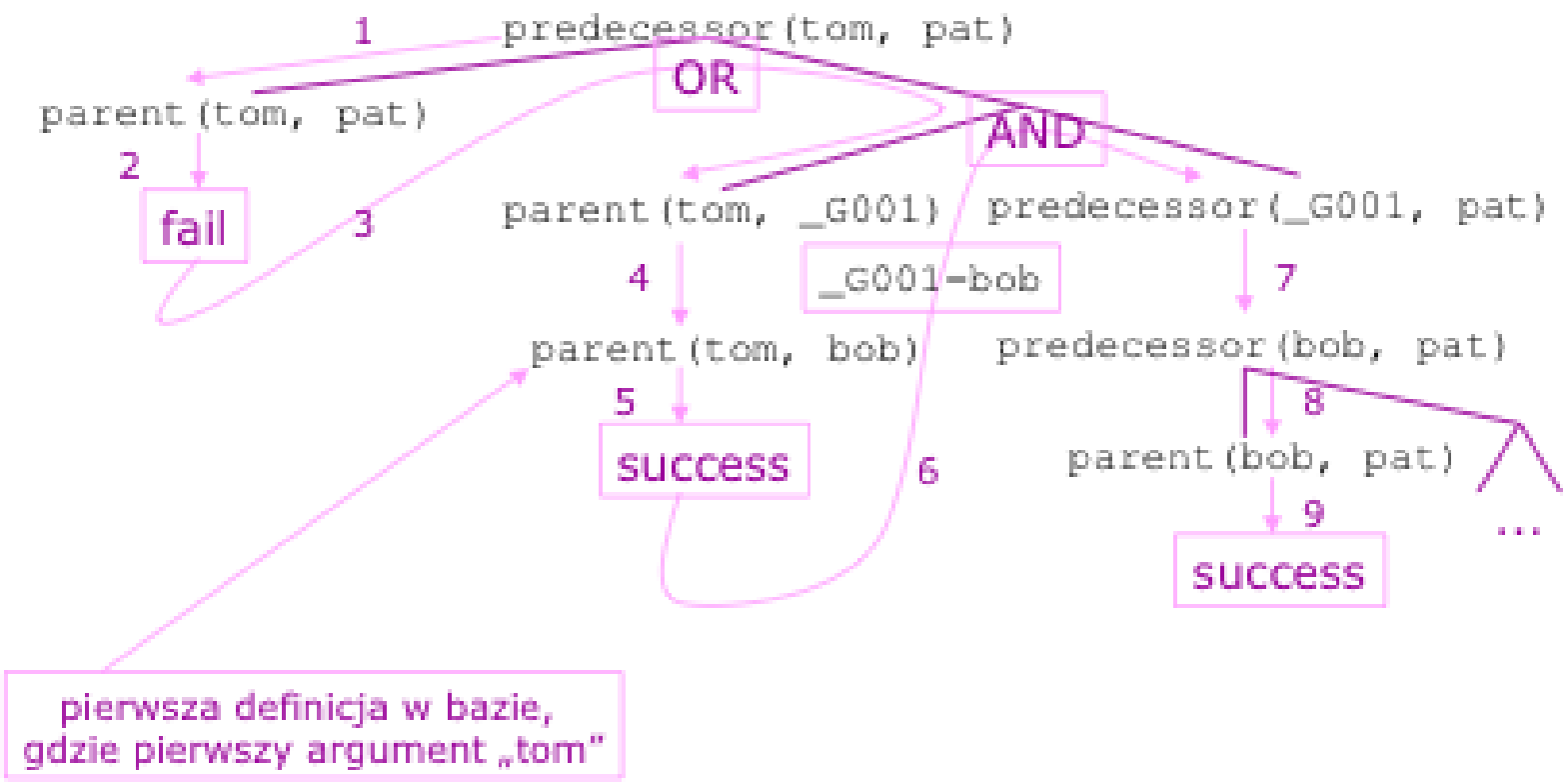
Rekurencja

Dany jest plik źródłowy

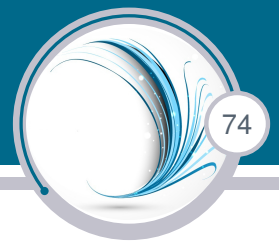
```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).
predecessor(X, Y):- parent(X, Y).
predecessor(X, Y):- parent(X, Z), predecessor(Z, Y).
```

Sprawdzamy, czy `predecessor(tom, pat).` jest prawdziwe.

Poprawna rekurencja



Rekurencja z błędem

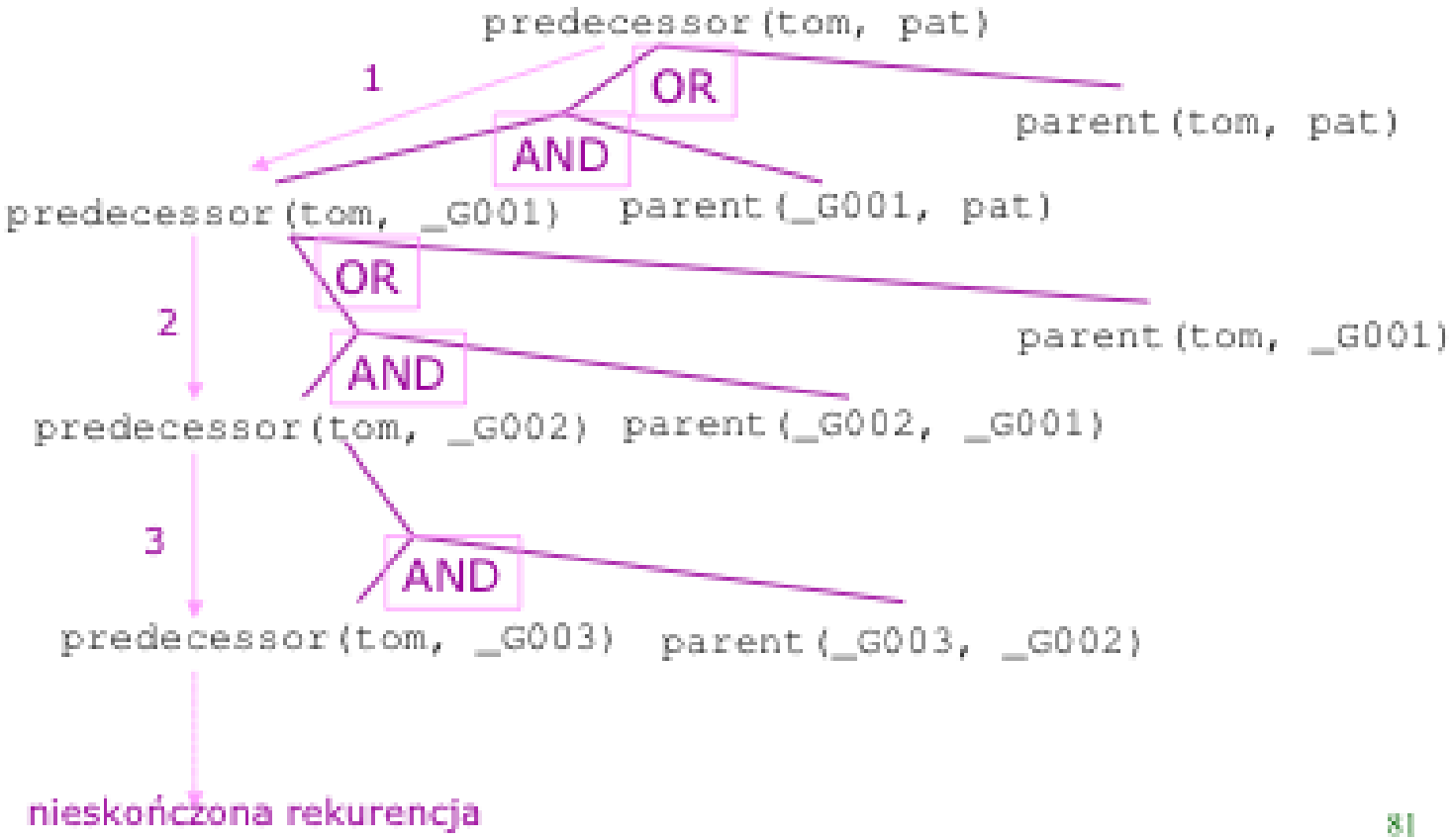


Zmieniony kod na przodka

```
predecessor(X, Y):- predecessor(X, Z), parent(Z, Y).
predecessor(X, Y):- parent(X, Y).
```

Sprawdzamy, czy `predecessor(tom, pat).` jest prawdziwe.

Niepoprawna rekurencja



81

Dziękuję za uwagę
Czas na pytania ????