Zarządzanie wiedzą

Wykład 7

Reprezentacja wiedzy w systemach inteligentnych

Rachunek predykatów pierwszego rzędu

Joanna Kołodziejczyk

06 maj 2011

Plan wykładu

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*'. *rodziec* 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 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

- Istnieja ludzie, którzy sa aniołami.
  $\exists x(czlowiek(x) \wedge aniol(x))$
- Istnieja ludzie, którzy nie sa aniołami.
  $\exists x(czlowiek(x) \wedge \neg aniol(x))$
- Nie tylko ludzie sa aniołami.
  $\exists x(\neg czlowiek(x) \wedge aniol(x))$

Mały kwantyfikator nie łączy się z implikacją!

Przykład na zdania z wielkim kwantyfikatorem

- Wszyscy Ludzie sa Aniołami
  $\forall x(czlowiek(x) \Rightarrow aniol(x))$
- Zaden Człowiek nie jest Aniołem
  $\forall x(czlowiek(x) \Rightarrow \neg aniol(x))$
- Tylko Ludzie sa Aniołami.
  $\forall x(czlowiek(x) \Rightarrow aniol(x))$

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:
  $\forall x, y\ person(x) \wedge ruler(y) \wedge loyalto(x, y) \Rightarrow trytoassassinate(x, y)$

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 zadnia.
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.

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

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

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

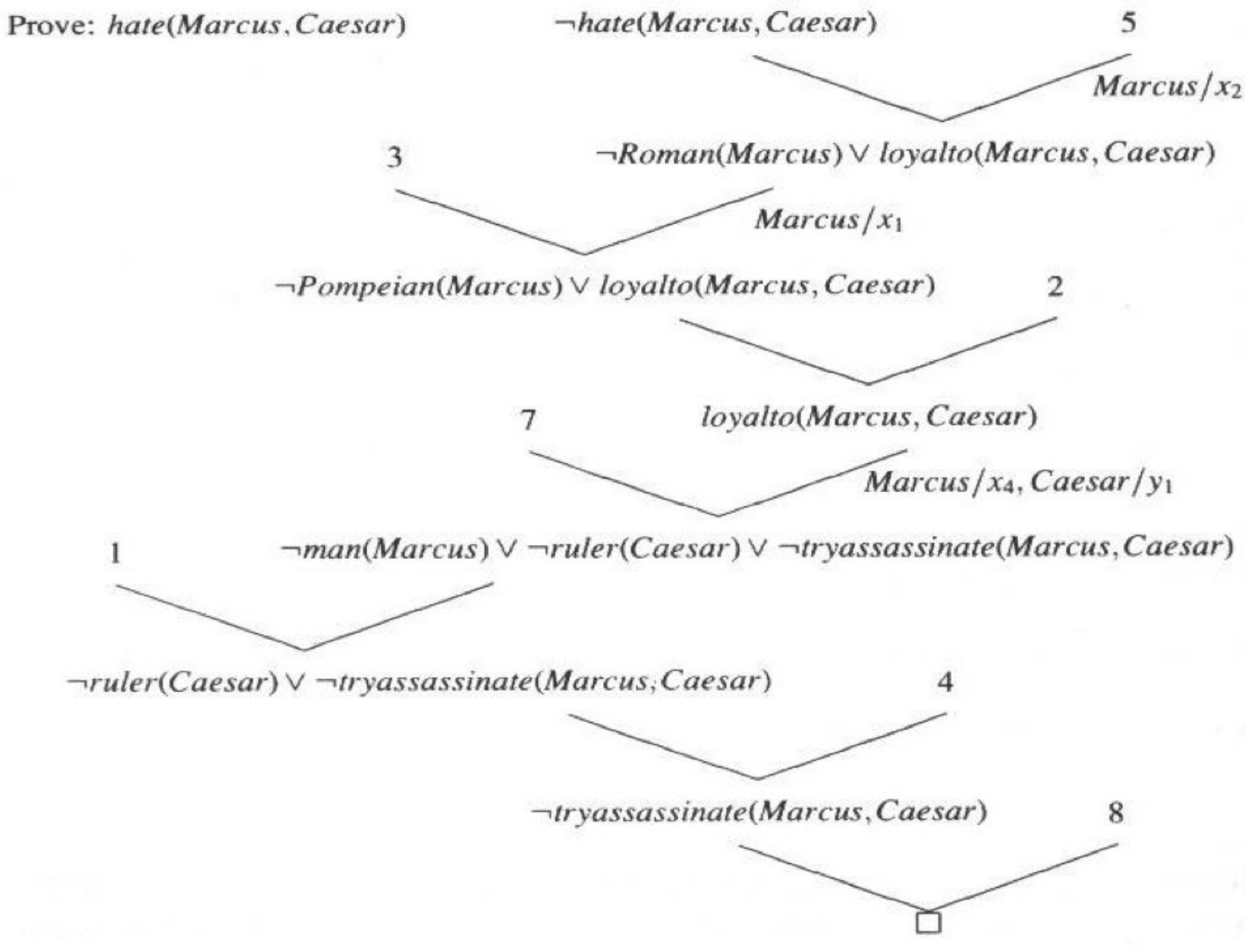
W wyniku konwersji bazy wiedzy ze slajdu ?? na postać CNF otrzymujemy:

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. $\neg man(x_4) \vee \neg ruler(y_1) \vee \neg tryassassinate(x_4, y_1) \vee loyalto(x_4, y_1)$
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



Plan wykładu

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) :- .

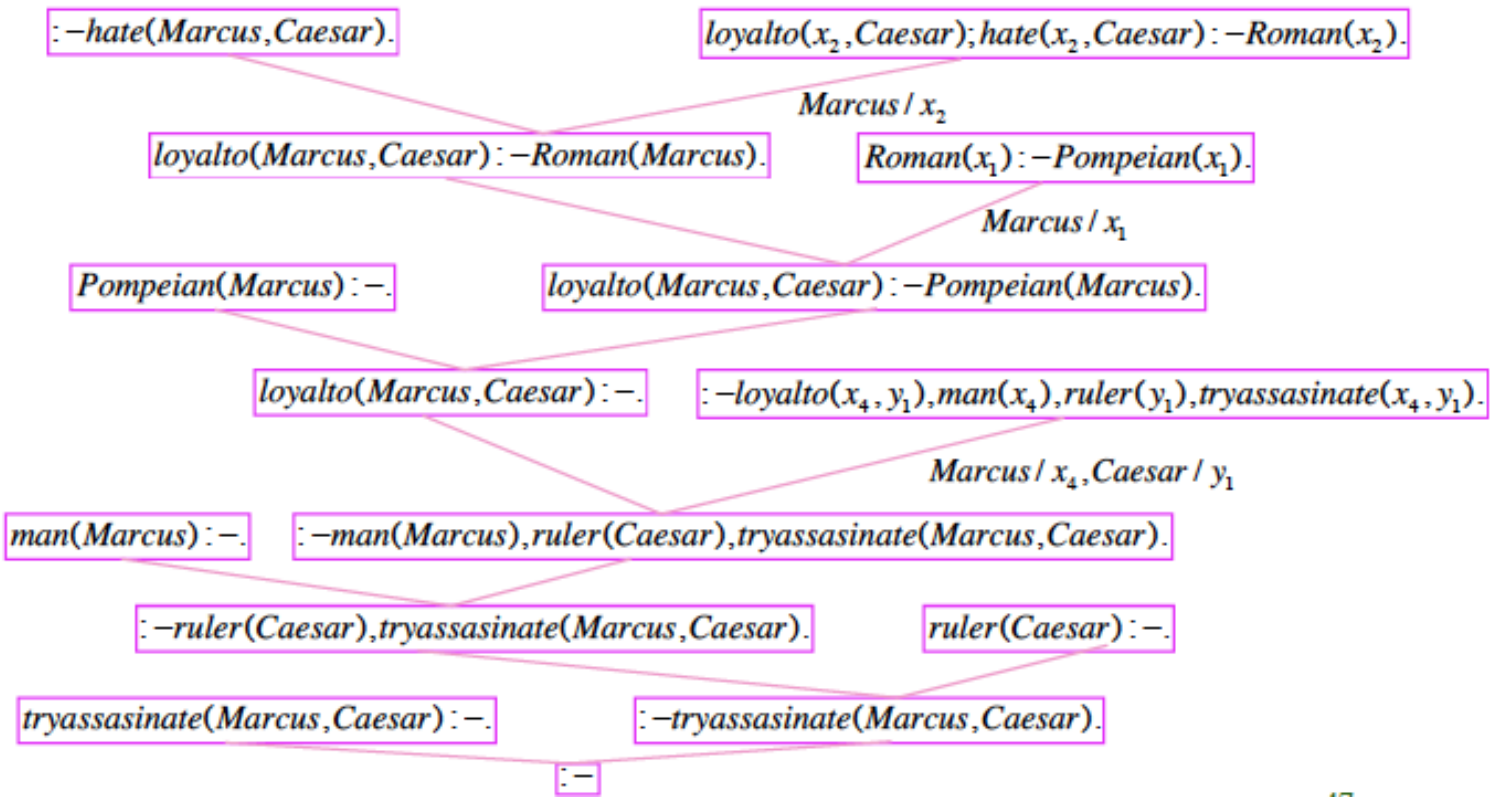
Przykład w nowej składni

W wyniku konwersji zdań CNF ze slajdu ?? 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
$\forall x pet(x) \land small(x) \Rightarrow apartment(x)$	`apartment(X) :- pet(X),small(X).`
$\forall x cat(x) \lor dog(x) \Rightarrow pet(x)$	`pet(X):-cat(X); dog(X).`
$\forall x poodle(x) \Rightarrow dog(x) \land small(x)$	`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.

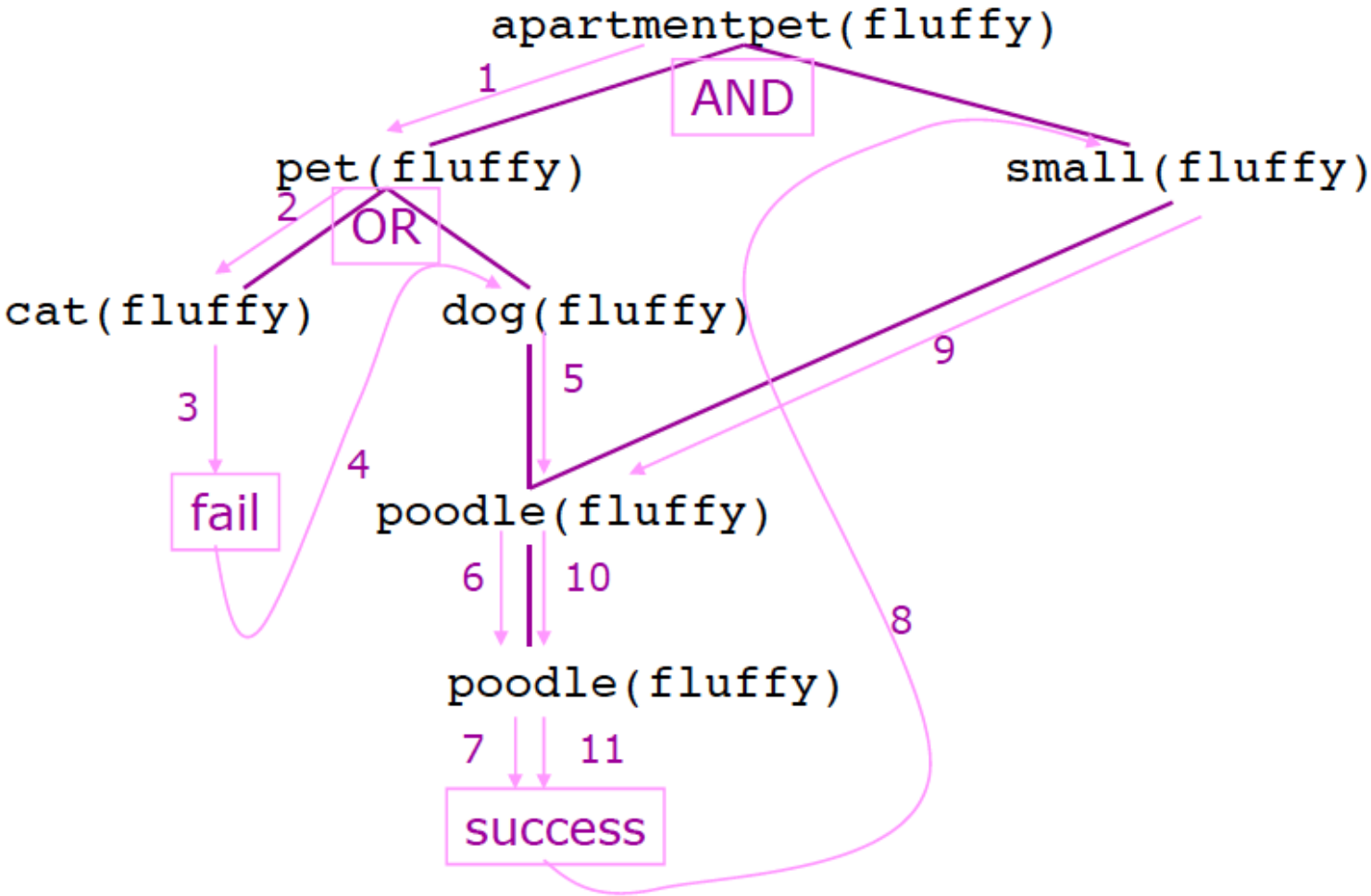
Kolejność wnioskowania

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.

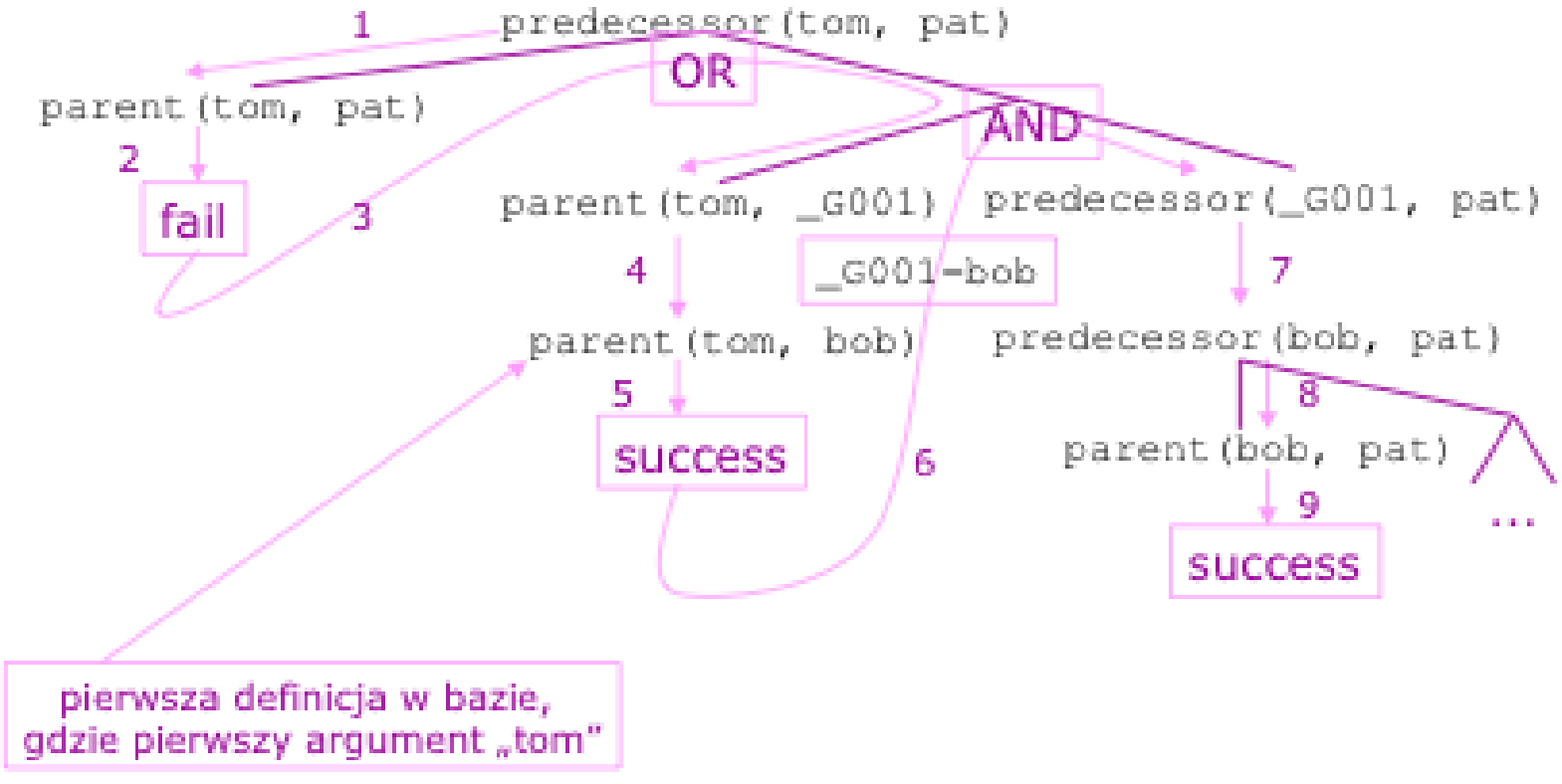
Rekurencja

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