Zarządzanie wiedzą

Reprezentacja wiedzy:
logika predykatów pierwszego rzędu

Plan wykładu

- Logika predykatów pierwszego rzędu
- Metoda rezolucji
- Unifikacja
- Przejście od logiki predykatów do Prologu
- PROgramm ing in LOGic
- Proceduraprzeglądania bazy wiedzy
- Predykaty rekurencyjne

Rachunku zdań kontra logika predykatów pierwszego rzędu

- Rachunek zdań:
  - Ograniczona ontologia – zakłada, że świat składa się tylko z faktów.
- Logika predykatów pierwszego rzędu
  - Może wyrażać fakty o wszystkich obiektach w mikroświecie. Pozwala na reprezentowanie ogólnych reguł rządzących rzeczywistością.
  - Dowolność reprezentacji faktów.
  - Uniwersalna: może wyrazić wszystko, co da się zaprogramować.

Składnia (syntaksa)

- Symbole stałe, np.: *A*, *B*, *Jan* – nazywają dokładnie jeden obiekt.
- Symbole predykatów, np.: *okrągły*, *brat* – szczególna relacja w modelu. *brat* może reprezentować relację „jest bratem”. *brat* jest symbolem predykatu binarnego, który zachowuje relację (lub nie) pomiędzy dwoma obiektami.
- symbole funkcji, np.: *cosinus*, *ojciec* – relacja funkcyjna jest zachowana dla dokładnie jednego obiektu w relacji. (Każdy ma jednego ojca, a każda wartość kąta ma dokładnie jedną wartość cosinusa).
- Wybór symboli zależy tylko od użytkownika!

Składnia – termy

- Term – wyrażenie logiczne odnoszące się do obiektu. Do termów zalicza się:
  - symbole stałe: Jan
  - symbole funkcji: lewa_noga(Jan) (jest to złożona nazwa – nie zwracamy wartości (lewej nogi Jana)!!).

Składnia – zdania atomowe

- Zdanie atomowe – wyraża fakt odnoszący się do obiektów i relacji pomiędzy nimi opisanymi symbolami predykatów.
  - brat(Ryszard, Jan)
  - małżeństwo(ojciec(Ryszard), matka(Jan))
- Zdanie atomowe jest *true*, jeżeli relacja opisana symbolem predykatu zachodzi pomiędzy obiektami podanymi jako argumenty relacji.

Składnia – zdania złożone

- Aby zbudować zdania złożone stosuje się operatory zdaniotwórcze (takie jak w rachunku zdań):
  - ¬ negacja (nie), np.: ¬ brat(Robin, Jan) (jest *true*, jeżeli Robin nie jest bratem Jana);
  - ∧ koniunkcja (i), np.: brat(Ryszard, Jan) ∧ brat(Jan, Ryszard) (jest *true*, jeżeli Ryszard jest bratem Jana i Jan jest bratem Ryszarda);
  - ∨ alternatywa (lub), np..: starszy(Jan, 30) ∨ młodszy(Jan, 30) (jest *true*, jeżeli Jan jest starszy niż 30 lub Jan ma mniej niż 30 lat);
  - → implikacja (z tego wynika, że) , np..: starszy(Jan, 30) → ¬ młodszy(Jan, 30) (Jeżeli Jan ma więcej niż 30 lat to nie może mieć mniej niż 30);

Składnia – Kwantyfikator wielki (ogólny)

- Pozwala określić własności grupy obiektów,
  - np..: Wszystkie koty to ssaki $\forall x: kot(x) \rightarrow ssak(x)$
- Stosuje się zmienne np.. x, y (oznaczone małymi literami).
- Zmienna jest termem.
- Co jeżeli poprzednik implikacji fałszywy? Nie ma to znaczenia, bo kwantyfikator ogranicza grupę obiektów do takich, dla których poprzednik jest tylko prawdziwy.
- Błąd w interpretacji: $\forall x: kot(x) \wedge ssak(x)$ (Wszystko jest kotami i wszystko jest ssakami). (Nie łączyć $\forall$ i $\wedge$!!!).

Składnia – Kwantyfikator mały (szczególny)

- Pozwala określić własności jakiegoś szczególnego obiektu bez nazywania go,
  - np.: Spot ma siostrę, która jest kotem
    $\exists x: siostra(x, Spot) \wedge kot(x)$
- Błąd w interpretacji: $\exists x: siostra(x, Spot) \rightarrow kot(x)$ (Nie łączyć $\exists$ i $\rightarrow$), bo rozwinięcie dla przypadków:

  siostra(Spot, Spot) $\rightarrow$ kot(Spot)
  - zadanie z implikacją będzie zawsze prawdziwe, jeżeli przesłanka jest fałszywa, więc dla każdego x, dla którego siostra(x, Spot) jest fałszywe. W przypadku iloczynu obie relacje muszą być prawdziwe.

Zagnieżdżanie kwantyfikatorów

- $\forall x\ \forall y$ jest równoważne $\forall y\ \forall x$, jest równoważne $\forall x, y$
- $\exists x\ \exists y$ jest równoważne $\exists y\ \exists x$

- $\exists x\ \forall y$ nie jest równoważne $\forall y\ \exists x$
- $\forall x\ \exists y$ kocha(x,y) „Każdy kogoś kocha”
- $\exists x\ \forall y$ kocha(x,y) „Istnieje ktoś, kto wszystkich kocha”
- $\exists y\ \forall x$ kocha(x,y) „Istnieje ktoś, kto jest przez wszystkich kochany”

- $\forall x$ [kot(x) $\vee$ $\exists x$ brat(Ryszard,x)] – w takim przypadku znaczenie ma najbardziej wewnętrzny kwantyfikator (w niniejszym przypadku $\forall$ nie ma żadnego efektu)

- Każda zmienna musi być związana z jakimś kwantyfikatorem.

Dualizm kwantyfikatorów

- $\forall$x lubi(x,Lody) $\quad\quad$ $\neg\exists$x $\neg$lubi(x,Lody)

  „Każdy lubi lody” $\quad$ „Nie istnieje ktoś, kto nie lubi lodów”

- $\exists$x lubi(x,Brokuły) $\quad\quad$ $\neg\forall$x $\neg$lubi(x,Brokuły)

  „Istnieje ktoś, kto lubi brokuły”

  „Nie każdy nie lubi brokuły”

- Prawa De Morgana:

$$\neg\,\forall\, x : P(x) \equiv \exists\, x : \neg\, P(x)$$

$$\forall\, x : \neg\, P(x) \equiv \neg\,\exists\, x : P(x)$$

$$\forall\, x : P(x) \equiv \neg\,\exists\, x : \neg\, P(x)$$

$$\exists\, x : P(x) \equiv \neg\,\forall\, x : \neg\, P(x)$$

Równość

- Symbol równości może służyć do budowania zdań.
- $term_1 = term_2$ jest prawdziwe w danej interpretacji jeżeli $term_1$ *i* $term_2$ odnoszą się do tego samego obiektu.
- np.. ojciec(Jan) = Henryk
- np.. $\forall x,y$ *rodzeństwo(x,y)* $\Leftrightarrow [\neg(x = y) \wedge \exists m,f \neg (m = f) \wedge$ rodzic(m,x) $\wedge$ rodzic(f,x) $\wedge$ rodzic(m,y) $\wedge$ rodzic(f,y)]

Logika predykatów na przykładzie

1. Marcus był człowiekiem.
2. Marcus był mieszkańcem Pompei.
3. Wszyscy mieszkańcy Pompei byli Rzymianami.
4. Cezar był władcą.
5. Wszyscy Rzymianie byli, albo lojalni wobec Cezara, lub nienawidzili go.
6. Każdy człowiek jest wobec kogoś lojalny.
7. Ludzie starają się zgładzić tylko takiego władcę, wobec którego nie są lojalni.
8. Marcus próbował zgładzić Cezara.

Logika predykatów na przykładzie

$man(Marcus)$

$Pompeian(Marcus)$

$\forall\, x : Pompeian(x) \rightarrow\ \ Roman(x)$

$ruler(Caesar)$

$\forall\, x : Roman(x) \rightarrow\ \ loyalto(x, Caesar) \vee\ \ hate(x, Caesar)$

$\forall\, x : \exists\, y : loyalto(x, y)$

$\forall\, x : \forall\, y : person(x) \wedge\ \ ruler(y) \wedge\ \ tryassass(x, y) \rightarrow\ \ \neg\ loyalto(x, y)$

$tryassass(Marcus, Caesar)$

Co pominięto w przykładzie

- Ad 1. Nie uwzględniono czasu przeszłego. W dalszych rozważaniach nie będzie to miało znaczenia.
- Ad 4. Imiona nie zawsze wskazują na konkretne indywiduum. Dokładność wymaga sporej ilości wiedzy przechowywanej w bazie.
- Ad 6. 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?
- Ad 7. Zdanie jest dwuznaczne. Interpretacja przedstawiona:

$$\forall x : \forall y : person(x) \wedge ruler(y) \wedge tryassass(x,y) \rightarrow \neg loyalto(x,y)$$

  - Są ludzie chcący zgładzić władcę zatem są wobec władcy nielojalni.
  - Inna interpretacja: Ludzie chcą zgładzenia władcy, wobec którego są nielojalni.

$$\forall x : \forall y : person(x) \wedge ruler(y) \wedge \neg loyalto(x,y) \rightarrow tryassass(x,y)$$

- Ad 7. Predykat: „try to assassinate” jest uproszczoną reprezentacją faktu. Predykat dzięki temu ma dwa argumenty.

Inżynieria wiedzy w logice predykatów pierwszego rzędu

1. Zidentyfikuj zadnie.
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 predykatów

- Dla rachunku zdań przedstawione zostały reguły wnioskowania takie jak modus ponens, rezolucji czy eliminacja AND i te reguły są poprawne dla logiki predykatów.
- Wymaga się jednak dodatkowych reguł, które poradzą sobie z kwantyfikatorami.

Unifikacja

- Algorytm unifikacji jest rekurencyjną procedurą, porównującą dwa termy i odkrywającą, czy istnieje zbiór podstawień, które sprawią, że termy staną się identyczne.
- Np. man(John) i man(John) unifikują się, natomiast man(John) i man(Spot) nie, bo mają różne argumenty.

Przykłady unifikacji

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}

Przykład unifikacji

- **dane: hate(x,y), hate(Marcus,z)**
- Unifikacja może zwrócić listę następujących 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)

  jeżeli dane: hate(Marcus,Paulus), hate(Marcus,Julian), hate(Marcus,Caesar)
- Podstawienia 1 i 2 są bardziej ogólne niż 3 i 4. Ostatecznie do dalszej rezolucji poszukuje się jak najbardziej ogólnego unifikatora (mgu).

Rezolucja

- Wersja reguły dla logiki predykatów pierwszego rzędu

$$\frac{l_1 \vee \cdots \vee l_k, \qquad m_1 \vee \cdots \vee m_n}{(l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

gdzie unifikacja(l_i, $\neg m_j$) = θ.

- Regułę stosuję się dla zdań w postaci formy CNF
- Przykład,

$$\frac{\neg Rich(x) \vee Unhappy(x) \qquad Rich(Ken)}{Unhappy(Ken)}$$

z θ = {Ken/x}

- Stosuje się rezolucję do zdań w CNF dla (KB $\wedge$ $\neg$α); gdzie α hipoteza (cel)

Konwersja do postaci normalnej koniunkcyjnej

- CNF - Conjunctive Normal Form.
- Postać ta to iloczyn sum literałów lub ich negacji
- Sumy rozdzielone iloczynami w postaci normalnej koniunkcyjnej to **klauzule**
- Cel: uzyskuje się mniej zagnieżdżonych komponentów – forma jest prostsza.

Przykład postaci normalnej

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

zdanie logiczne

$$\forall x : [Roman(x) \wedge know(x, Marcus)] \rightarrow$$
$$[hate(x, Caesar) \vee (\forall y : \exists z : hate(y, z) \rightarrow thinkcrazy(x, y))]$$

klauzula

$$\neg Roman(x) \vee \neg know(x, Marcus) \vee$$
$$hate(x, Caesar) \vee \neg hate(y, z) \vee thinkcrazy(x, y)$$

Algorytm konwersji do klauzul

- Algorytm składa się z dziewięciu kroków.
- Operuje na złożonym zdaniu logicznym i poprzez przeprowadzanie upraszczających operacji doprowadzi do powstania uproszczonej w formie klauzuli.
- Zdanie i jego postać CNF są logicznymi ekwiwalentami.

Etap 1 – Usunięcie implikacji

- Korzysta z własności

$$a \rightarrow b \equiv \neg a \vee b$$

- Przykład:

$$\forall x : male(X) \rightarrow man(X)$$

$$\forall x : \neg male(X) \vee man(X)$$

Etap 2 – Przesunięcie negacji

- Krok ten prowadzi do uproszczenia wyrażeń z negacją.
- Redukuje nawiasy.
- Korzysta z własności

$$\neg(\neg p) \equiv p$$

$$\neg(a \wedge b) \equiv \neg a \vee \neg b$$

$$\neg(a \vee b) \equiv \neg a \wedge \neg b$$

$$\neg \forall x : P(x) \equiv \exists x : \neg P(x)$$

$$\neg \exists x : P(x) \equiv \forall x : \neg P(x)$$

Etap 3 – standaryzacja zmiennych

- Zmienne opisywane przez różne kwantyfikatory otrzymują różne nazwy.
- Przed:

$$\forall x : P(x) \vee \forall x : Q(x)$$

- Po:

$$\forall x : P(x) \vee \forall y : Q(y)$$

Etap 4 – przesunięcie wszystkich kwantyfikatorów

- Przenosi się wszystkie kwantyfikatory poza wyrażenie. Nie powoduje to zmiany interpretacji wyrażenia.
- Przed:

$$\forall x : (\neg\, mezczyzna(x) \vee\ (\forall y : (\neg\, kobieta(y) \vee\ lubi(x, y))))$$

- Po:

$$\forall x : \forall y : (\neg\, mezczyzna(x) \vee\ (\neg\, kobieta(y) \vee\ lubi(x, y)))$$

Etap 5 – skolemizacja

- Usunięcie kwantyfikatora istnienia.
- Wprowadza się nowe symbole reprezentujące stałe, w miejsce zmiennych spod usuwanych kwantyfikatorów.
- Przypadek pierwszy:
  - Przed: $\exists x : President(x)$
  - Po: $President(s1)$
- Przypadek z wielkim kwantyfikatorem (S2(x) – funkcja od zmiennej x):
  - Przed: $\forall x : \exists y : father(x, y)$
  - Po: $\forall x : father(x, S2(x))$

Etap 6 – usunięcie wielkiego kwantyfikatora

- Opuszcza się w wyrażeniu kwantyfikatory wielkie i nie zmienia to znaczenia całego wyrażenia.
- Przed:

$$\forall\, x : father(x, mark)$$

- Po:

$$father(x, marek)$$

Etap 7 – wyprowadzenie i przed lub

- Konwersja wyrażenia na iloczyn sum. Koniunkcje nie mogą pojawiać się wewnątrz alternatyw.
- Przed:

$$(p(x) \wedge \neg q(x)) \vee r(y)$$

- Po:

$$(p(x) \vee r(y)) \wedge (\neg q(x) \vee r(y))$$

Etap 8 – stworzenie klauzul

- Stworzenie niezależnych klauzul dla każdej koniunkcji. Każdy czynnik iloczynu musi mieć wartość logiczną 1.
- Przed: $(p(x) \vee \neg q(x)) \wedge (\neg q(x) \vee r(y))$

- Po: powstały 2 klauzule

$$1.(p(x) \vee r(y))$$

$$2.(\neg q(x) \vee r(y))$$

Etap 9 – standaryzacja zmiennych

- Standaryzacja zmiennych w niezależnych klauzulach.
- Każdej zmiennej można nadać inną nazwę, co nie zmieni znaczenia logicznego, a zapobiega myleniu zmiennych podczas dowodzenia.

Przykład konwersji

$$\forall x : [Roman(x) \wedge know(x, Marcus)] \rightarrow [hate(x, Caesar) \vee (\forall y : \exists z : hate(y, z) \rightarrow thinkcrazy(x, y))]$$

↓ (etap 1)

$$\forall x : \neg [Roman(x) \wedge know(x, Marcus)] \vee [hate(x, Caesar) \vee (\forall y : \neg (\exists z : hate(y, z)) \vee thinkcrazy(x, y))]$$

↓ (etap 2)

$$\forall x : [\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caesar) \vee (\forall y : \forall z : \neg hate(y, z) \vee thinkcrazy(x, y))]$$

↓ (etap 4)

$$\forall x : \forall y : \forall z : [\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caesar) \vee (\neg hate(y, z) \vee thinkcrazy(x, y))]$$

↓ (etap 6)

$$[\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caesar) \vee (\neg hate(y, z) \vee thinkcrazy(x, y))]$$

↓ (etap 7)

$$\neg Roman(x) \vee \neg know(x, Marcus) \vee hate(x, Caesar) \vee \neg hate(y, z) \vee thinkcrazy(x, y)$$

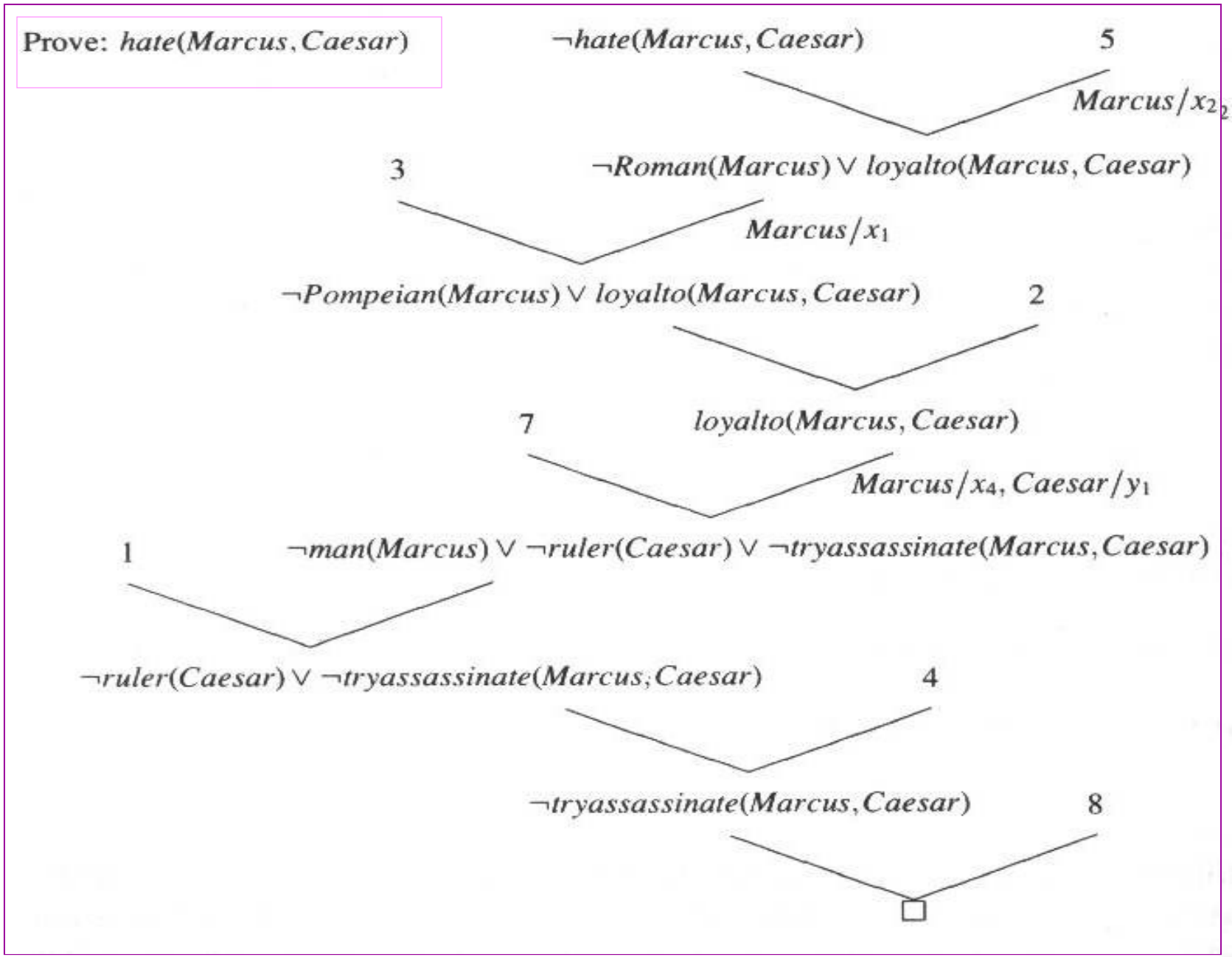
Rezolucja w logice predykatów

1. Konwersja aksjomatów **F** na klauzule.
2. Zaneguj **P** (predykat do udowodnienia) i przekształć na klauzulę. Dodaj do zbioru klauzul **F**.
3. Powtarzaj, do osiągnięcia klauzuli pustej, lub zatrzymaj, gdy nie ma postępu:
   a. Wybierz **2 klauzule** (rodzicielskie).
   b. Porównaj je. Klauzula wynikowa - ***rezolwenta*** jest sumą logiczną wszystkich literałów obu klauzuli rodzicielskich z zastosowaniem odpowiednich podstawień z następującymi wyjątkami: Jeżeli istnieje para literałów **T1** i ¬**T2**, takich że jedna z klauzul rodzicielskich zawiera **T1**, a druga zwiera **T2**, to zarówno **T1** jak i **T2** są unifikowane. **T1** i **T2** 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.
   c. Jeżeli ***rezolwenta*** jest klauzulą pustą, to dowód zkończono. Jeżeli nie, to dodaj klauzulę do zbioru klauzul.

Przykład 1

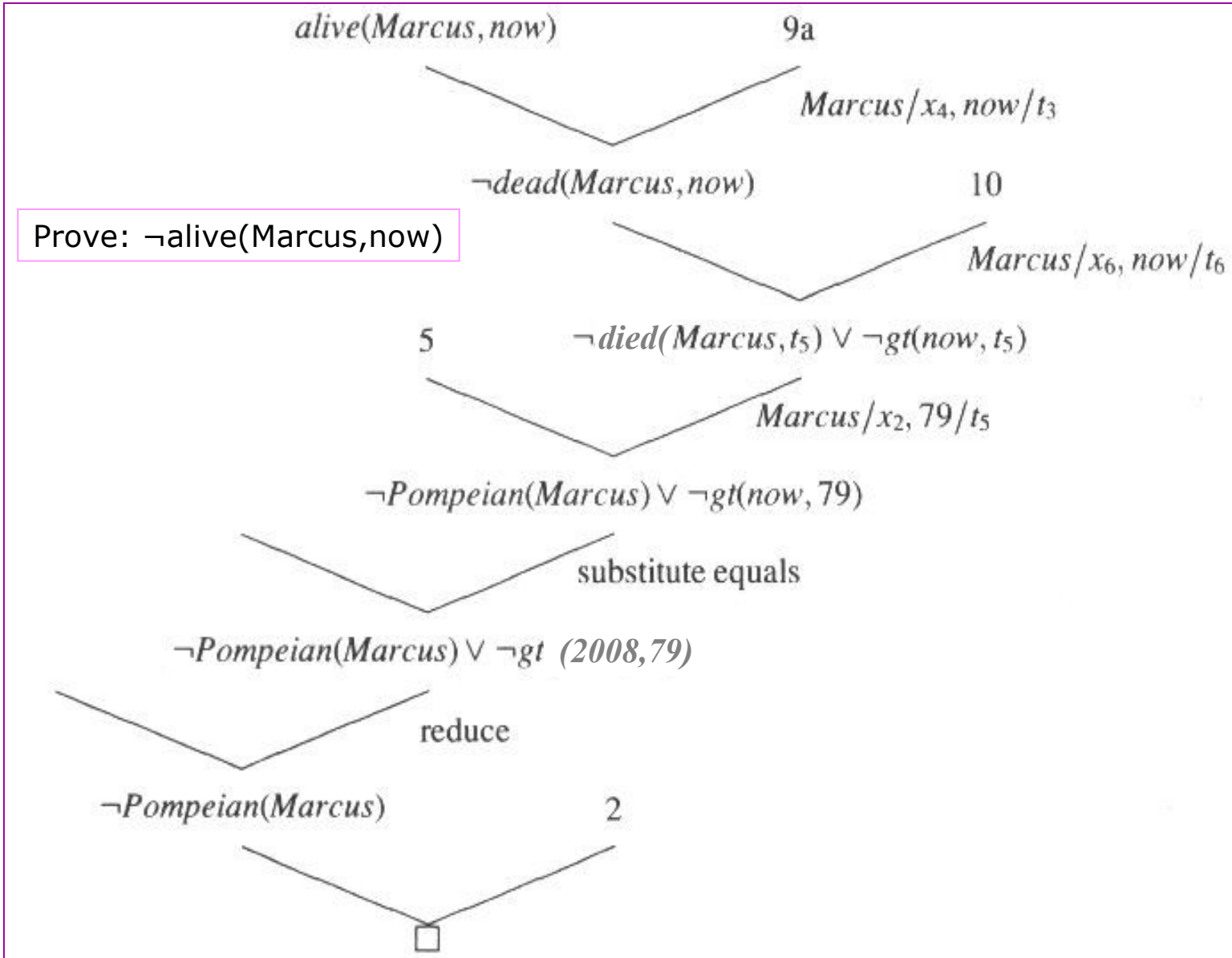
- Przekształcenie zdań ze slajdu 15 na klauzule:

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. $loyalto(x_3, f1(x_3))$
7. $\neg\ loyalto(x_4, y_1) \vee\ \neg\ man(x_4) \vee\ \neg\ ruler(y_1) \vee\ \neg\ tryassasinate(x_4, y_1)$
8. $tryassasinate(Marcus, Caesar)$


Prove: hate(Marcus, Caesar)
¬hate(Marcus, Caesar)
5
Marcus/x2
¬Roman(Marcus) ∨ loyalto(Marcus, Caesar)
3
Marcus/x1
¬Pompeian(Marcus) ∨ loyalto(Marcus, Caesar)
2
loyalto(Marcus, Caesar)
7
Marcus/x4, Caesar/y1
¬man(Marcus) ∨ ¬ruler(Caesar) ∨ ¬tryassassinate(Marcus, Caesar)
1
¬ruler(Caesar) ∨ ¬tryassassinate(Marcus, Caesar)
4
¬tryassassinate(Marcus, Caesar)
8
□

Przykład 2 (z funkcjami)

1. man(Marcus)
2. Pompeian(Marcus)
3. born(Marcus,40)
1. ¬man(x1)∨mortal(x1)
2. ¬Pompeian(x2)∨died(x2,79)
1. erupted(volcano,79)
1. ¬mortal(x3)∨ ¬ born(x3,t1) ∨ ¬gt(t2-t1,150) ∨dead(x3,t2)
1. now=2008
2. Predykat 9a i 9b
   - ➔ ¬ alive(x4,t3)∨ ¬dead(x4,t3)
   - ➔ dead(x5,t4) ∨alive(x5,t4)
1. ¬died(x6,t5) ∨ ¬gt(t6,t5) ∨dead(x6,t6)


alive(Marcus, now)
9a
Marcus/x4, now/t3
¬dead(Marcus, now)
10
Prove: ¬alive(Marcus,now)
Marcus/x6, now/t6
¬died(Marcus, t5) ∨ ¬gt(now, t5)
5
Marcus/x2, 79/t5
¬Pompeian(Marcus) ∨ ¬gt(now, 79)
substitute equals
¬Pompeian(Marcus) ∨ ¬gt (2008,79)
reduce
¬Pompeian(Marcus)
2

Co wiemy

- Czym jest logika predykatów.
- Jak działa algorytm rezolucji.
  - Jeżeli teza {**A1, A2,..., An**} jest niesprzeczna, formuła **B** jest wnioskiem z {**A1, A2, ..., An**} wtedy i tylko wtedy gdy teza {**A1, A2, ..., ¬B**} jest sprzeczna.
- Czym jest postać klauzulowa.
- Czym jest unifikacja.

Interpretacja klauzul

- Zakładamy klauzulę

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

- Co jest równoważne:

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

- Co jest równoważne:

$$(p_1(x) \vee p_2(x) \vee \ldots \vee p_n(x)) \leftarrow (q_1(x) \wedge q_2(x) \wedge \ldots \wedge q_n(x))$$

- Oznaczamy $\wedge$ jako `,` , $\vee$ jako `;` , a $\leftarrow$ jako `:-` .

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

Przykład

- Każdego małego zwierzaka można trzymać w mieszkaniu.

$$\forall x : maly(x) \wedge zwierzak(x) \rightarrow trzymac(x, mieszkanie)$$

- Po przekształceniu na klauzule:

$$\forall x : \neg (maly(x) \wedge zwierzak(x)) \vee trzymac(x, mieszkanie)$$
$$\forall x : \neg maly(x) \vee \neg zwierzak(x) \vee trzymac(x, mieszkanie)$$
$$\neg maly(x) \vee \neg zwierzak(x) \vee trzymac(x, mieszkanie)$$

- Co daje: $trzymac(x, mieszkanie) \vee (\neg (maly(x) \wedge zwierzak(x)))$

$$trzymac(x, mieszkanie) \leftarrow maly(x) \wedge zwierzak(x)$$
$$trzymac(x, mieszkanie) :- maly(x), zwierzak(x)$$

Reguły i Fakty

- 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) :- .

Klauzule – slajd15

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. $loyalto(x_3, f1(x_3))$

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

8. $tryassasinate(Marcus, Caesar)$

Te same klauzule w nowej składni

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. $loyalto(x_3, f1(x_3)) :- .$

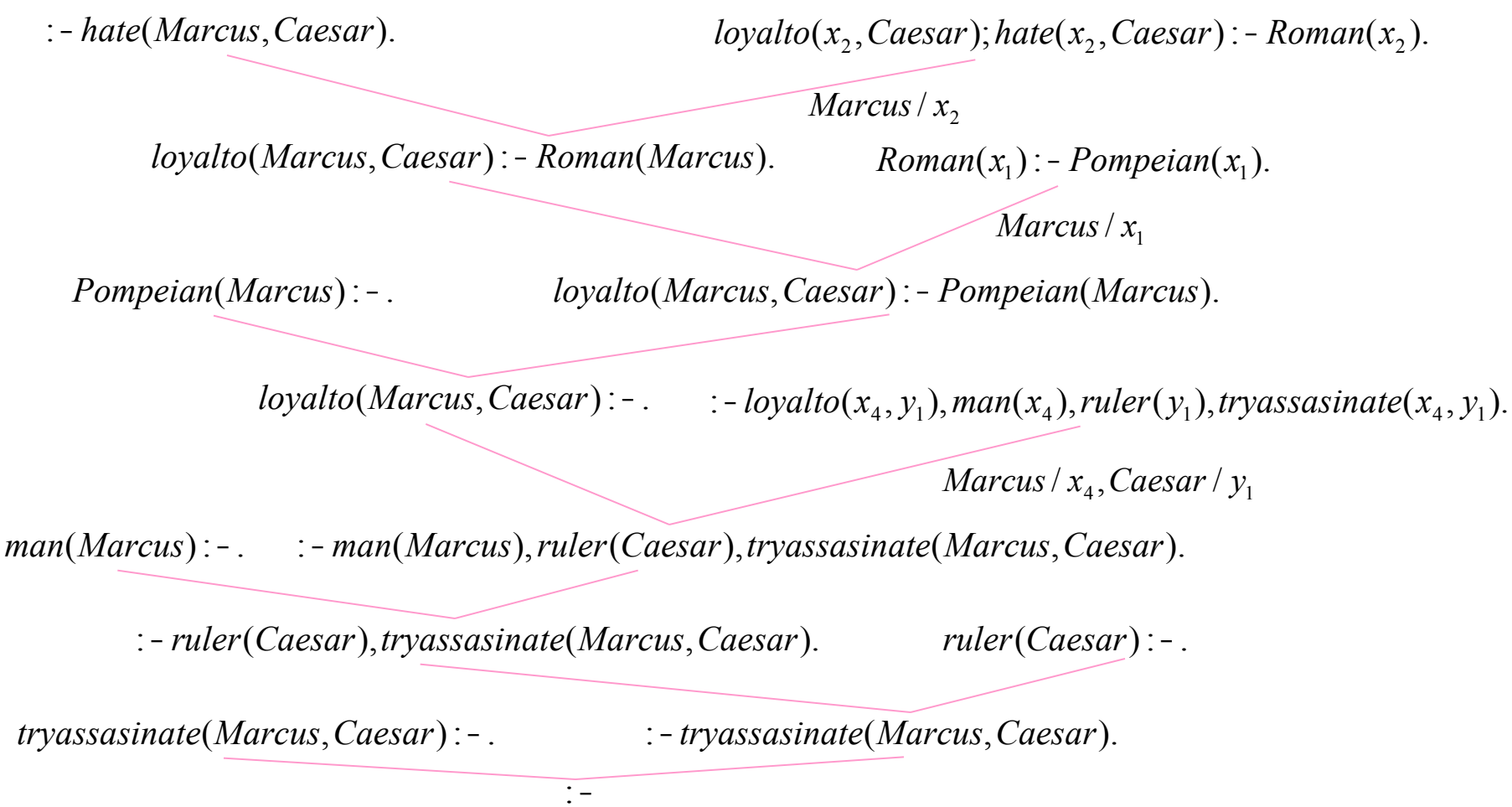
7. $:- loyalto(x_4, y_1), man(x_4), ruler(y_1), tryassasinate(x_4, y_1).$

8. $tryassasinate(Marcus, Caesar) :- .$

Przykład dowodzenia rezolucją – założenia

- Chcemy udowodnić, że Marcus nienawidzi Caesara.
- Zatem stawiamy hipotezę hate(Marcus, Caesar).
- Spełniając założenia rezolucji negujemy hipotezę. Co daje klauzulę: ¬hate(Marcus, Caesar).
- W nowej składni klauzula będzie miała postać: :- hate(Marcus, Caesar).

Przykład cd.


:- hate(Marcus, Caesar).
loyalto(x2, Caesar); hate(x2, Caesar) :- Roman(x2).
Marcus / x2
loyalto(Marcus, Caesar) :- Roman(Marcus).
Roman(x1) :- Pompeian(x1).
Marcus / x1
Pompeian(Marcus) :- .
loyalto(Marcus, Caesar) :- Pompeian(Marcus).
loyalto(Marcus, Caesar) :- .
:- loyalto(x4, y1), man(x4), ruler(y1), tryassasinate(x4, y1).
Marcus / x4, Caesar / y1
man(Marcus) :- .
:- man(Marcus), ruler(Caesar), tryassasinate(Marcus, Caesar).
:- ruler(Caesar), tryassasinate(Marcus, Caesar).
ruler(Caesar) :- .
tryassasinate(Marcus, Caesar) :- .
:- tryassasinate(Marcus, Caesar).
:-

Jeszcze raz rezolucja

- Zupełność: jeżeli jakiś fakt jest wnioskiem z hipotezy, możliwe jest udowodnienie go przez wykazanie fałszywości zbioru przesłanek uzupełnionych negacją faktu.
- Jak zmyślnie dobierać kolejne klauzule w celu dopasowania?
- Zadowoli nas system, który pokaże wszystkie możliwe wnioski.

Klauzula Horna

- Jest to klauzula zawierająca co najwyżej jeden niezanegowany predykat.
- Klauzulami Horna nie są zdania typu:

$$\forall x : Roman(x) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$$

$$loyalto(x_2, Caesar); hate(x_2, Caesar) :- Roman(x_2).$$

- To uproszczenie pozwala na ułatwienie wnioskowania.
- Klauzule Horna dzieli się na:
  - z głową: posiadające jeden niezanegowany predykat;np..
    man(x) :- person(x).
    man(Marcus) :-.
  - bez głowy: bez niezanegowanego predykatu;
    :-person(x).

Zbiory klauzul Horna jako baza wiedzy i zapytania

- Wszystkie klauzule poza jedną posiadają głowę.
- Założenie to pozwala na rozwiązanie dowolnego problemu.
- 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

- 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.

Implementacja

1. Stosując zasady logiki tworzy się system zawierający fakty i reguły.
2. Zadaje się pytania dotyczące wiedzy zebranej w punkcie 1.
   - ➔ pytania mogą mieć charakter „tak” lub „nie”, lub
   - ➔ wyszukaj wszystkie dane, dla których podane wyrażenie będzie miało wartość TRUE.

Przykład

Reprezentacja logiczna

1. $\forall x$: pet(x) $\wedge$ small(x) → apartmentpet(x)
2. $\forall x$: cat(x) $\vee$ dog(x) → pet(x)
3. $\forall x$: poodle(x) → dog(x) $\wedge$ small(x)
1. poodle(fluffy)

Reprezentacja w PROLOGU

```
1. apartmentpet(X) :-
   pet(X), small(X).
2. pet(X) :- cat(X).
3. pet(X) :- dog(X).
4. dog(X) :- poodle(X).
5. small(X) :-
   poodle(X).
6. poodle(fluffy).
```

reguły

fakt

Zależność Prologu i logiki

- 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).

Składnia

- . - koniec klauzuli;
- ; - OR (dysjunkcja);
- , - AND(koninkcja);
- :- - ← implikacja;
- % - komentarz;
- man(‘Marcus’). - fakt
- man(X) :- person(X). - reguła

Termy

- Każda klauzula składa się z termów:
  - *stałe* (atomy, identyfikator zaczyna się z małej litery lub liczby)
    - *np..* `pismo, 4.5`
  - *zmienne* (identyfikator zaczyna się z wielkiej litery lub podkreślenia)
    - *np..* `Zmienna, _zmienna`
  - *struktury* (predykaty lub struktury danych)
    - *np..* `matka(ala), data(Rok,Miesiac,Dzien)`

Niuanse

- Dlaczego:

```
1. pet(X) :- cat(X).
2. pet(X) :- dog(X).
```

- jest równoważne: `pet(X) :- cat(X);dog(X).`

$$pet(X) :- dog(X). \quad pet(X) :- cat(X). \rightarrow pet(X) \vee \neg dog(X) \quad pet(X) \vee \neg cat(X) \rightarrow (pet(X) \vee \neg dog(X)) \wedge (pet(X) \vee \neg cat(X))$$

$$pet(X) \vee (\neg dog(X) \wedge \neg cat(X)) \rightarrow pet(X) \vee (\neg (dog(X) \vee cat(X)))$$

$$pet(X) \leftarrow (dog(X) \vee cat(X)) \rightarrow pet(X) :- dog(X); cat(X).$$

Unifikacja

- Proces kojarzenia zmiennych i wartości. Zmienna, której przypisano stałą wartość nazywa się ukonkretnioną.
- Reguły unifikacji:
  - Stała może być zunifikowana tylko ze sobą.
  - Dwie struktury 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.

Unifikowalność

- Znak równości w Prologu oznacza operację unifikacji
- Na przykład:

```
?- jablko = jablko.
Yes
?- jablko = gruszka.
No
?- lubi(Osoba, jablko) = lubi(adam, Jedzenie).
Osoba = adam
Jedzenie = jablko ;
No
?- lubi(Osoba, gruszka) = lubi(adam, jablko).
No
?- lubi(Osoba, Jedzenie) = lubi(adam, Jedzenie).
Osoba = adam
Jedzenie = _G158 ;
No
```

Wewnętrzna reprezentacja niezunifikowanej zmiennej.

Jakiekolwiek podstawienie dowodzi prawdy

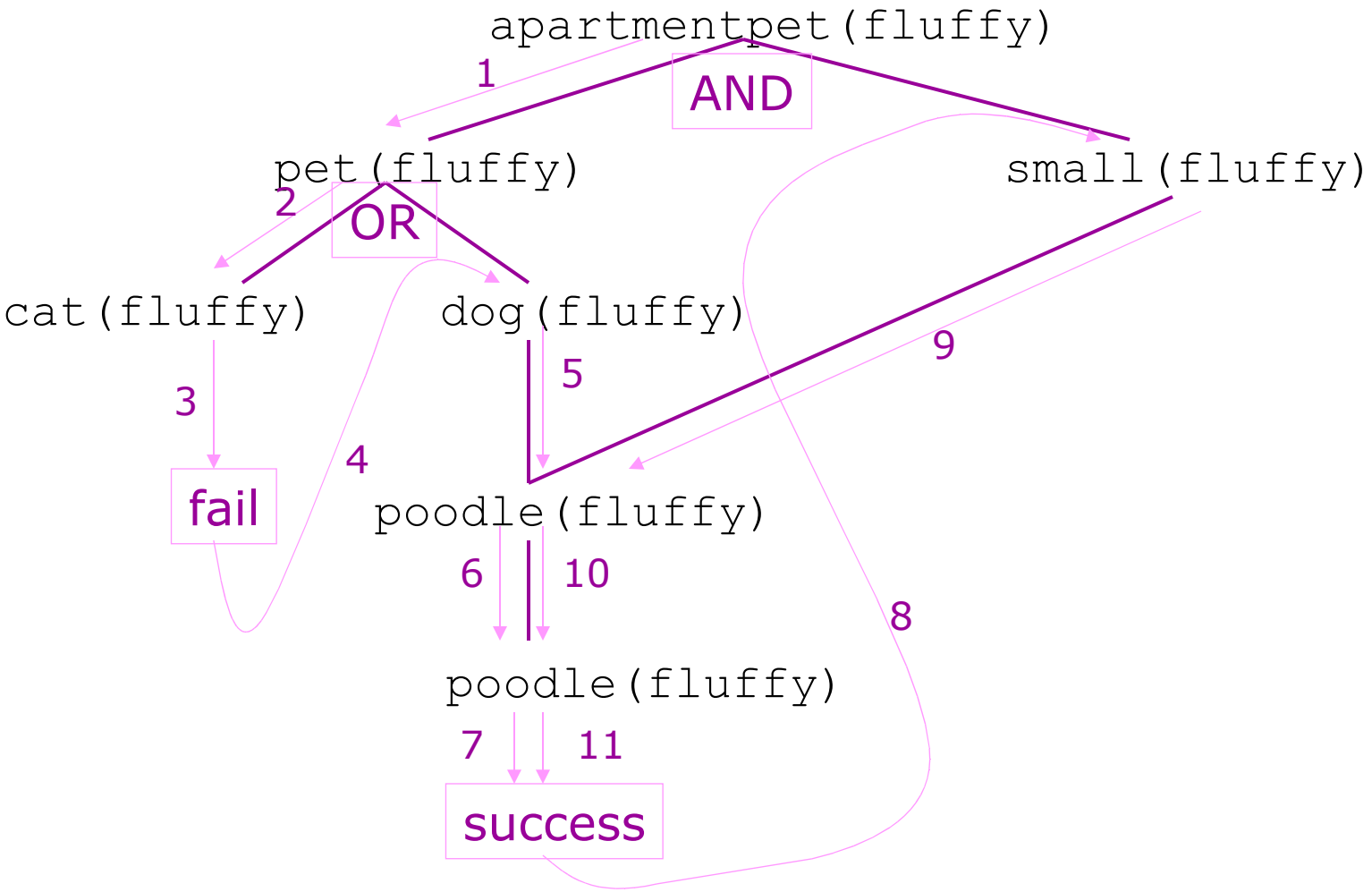
Kolejność wnioskowania

- Dany jest plik źródłowy:

```
1. apartmentpet(X) :- pet(X), small(X).
2. pet(X) :- cat(X).
3. pet(X) :- dog(X).
4. dog(X) :- poodle(X).
5. small(X) :- poodle(X).
6. poodle(fluffy).
```

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

Schemat odpowiedzi



Kolejność klauzul

- 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.