Modele obliczalności w logice

dr hab. inż. Joanna Józefowska, prof. PP

Język rachunku predykatów I rzędu
First Order Logic (FOL)

$A = Z \cup F \cup P \cup S \cup \{(,), ,\}$

A - alfabet symboli

Z - zmienne x_1, x_2, ...

F - n-argumentowe symbole funkcyjne

P - n-argumentowe symbole predykatowe

S - symbole logiczne $\{\neg, \wedge, \vee, \Leftrightarrow, \Rightarrow, \forall, \exists\}$

nad A budujemy wyrażenia: termy, formuły atomowe, formuły

Definicje

Zbiór termów T_m jest najmniejszym zbiorem spełniającym.

Każdy skończony ciąg znaków języka rachunku predykatów nazywamy wyrażeniem tego języka.

(i) Wszystkie zmienne indywiduowe x_i oraz wszystkie nazwy indywidualne a_i są termami (albo formułami nazwowymi) języka rachunku predykatów.

(ii) Jeżeli $\alpha_1, \alpha_2, ..., \alpha_n$ są dowolnymi termami, to wyrażenie $F_k^n(\alpha_1, \alpha_2, ..., \alpha_n)$ jest również termem.

(iii) Nie ma innych termów (języka rachunku predykatów) prócz zmiennych indywiduowych, nazw indywidualnych i tych termów, które można skonstruować wedle reguły (ii).

Definicje

Dane wyrażenie jest formułą zdaniową atomową wtedy i tylko wtedy, gdy ma ono postać $P_k^n(\alpha_1, \alpha_2, ..., \alpha_n)$, gdzie $\alpha_1, \alpha_2, ..., \alpha_n$ są dowolnymi termami.

Definicje

Zbiór formuł zdaniowych jest najmniejszym zbiorem spełniającym.

(i) Każda formuła zdaniowa atomowa jest formułą zdaniową.

(ii) Jeżeli A jest dowolną formułą zdaniową, to wyrażenia $\neg(A)$, $\forall x_i\,(A)$, $\exists x_i\,(A)$, są również formułami zdaniowymi.

(iii) Jeżeli A i B są dowolnymi formułami zdaniowymi, to wyrażenia $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \Leftrightarrow B$, są również formułami zdaniowymi.

(iv) Nie ma innych formuł zdaniowych (języka rachunku predykatów), prócz tych, które można utworzyć wedle reguł (i)-(iii).

Definicje

Wyrażenie A w dowolnej formule zdaniowej o postaci $\forall x_i\,(A)$ lub o postaci $\exists x_i\,(A)$ nazywamy zasięgiem odpowiedniego kwantyfikatora.

Zmienna x_i występująca na danym miejscu w formule zdaniowej A jest na tym miejscu związana, jeżeli jest ona podpisana pod którymś z kwantyfikatorów lub też znajduje się w zasięgu kwantyfikatora, pod którym podpisana jest również zmienna x_i.

Jeżeli zmienna x_i, występująca na danym miejscu w formule zdaniowej A nie jest na tym miejscu związana, to mówimy, że jest ona na tym miejscu wolna w A.

Definicje

Mówimy, że x_i jest zmienną wolną w A wtedy i tylko wtedy, gdy przynajmniej na jednym miejscu zmienna ta jest wolna w A.

Formuły zdaniowe nie zawierające żadnych zmiennych wolnych nazywamy zdaniami (języka rachunku predykatów).

Przykłady

Niech A = {X, +, 1, 2, =} $\cup$ S $\cup$ {(,), ,}

Termy:	X, 1, 2, X +1, lub +(X,1)
Formuły atomowe:	X = X, 1 + 2 = X (lub =(X, X)
Formuły zdaniowe:	$\forall$ X = X, X = 1 $\Rightarrow$ 1 = X X
Zdania:	1 + 2 = 2 + 1

Dopasowanie wyrażenia rachunku predykatów do wzorca

Mówimy, że term α jest podstawialny za zmienną x_i do formuły zdaniowej A wtedy i tylko wtedy, gdy zmienna x_i nie leży w A jako wolna w zasięgu żadnego kwantyfikatora wiążącego którąś ze zmiennych występujących w α.

Dane są dwa wyrażenia rachunku predykatów będące formułami atomowymi. Jedną z nich nazwijmy wzorcem.

Czy można znaleźć takie podstawienie termów za zmienne wzorca, w efekcie którego obydwa wyrażenia uzyskają identyczną postać?

Procedura „match”

φ – przed wywołaniem procedury jest zbiorem pustym

(A) Jeśli wzorzec i wyrażenie dopasowywane są identyczne, to koniec - wynikiem jest znalezione dotychczas podstawienie φ,

(B) w przeciwnym przypadku, niech $\langle t_1, t_2 \rangle$ będzie parą niezgodną,

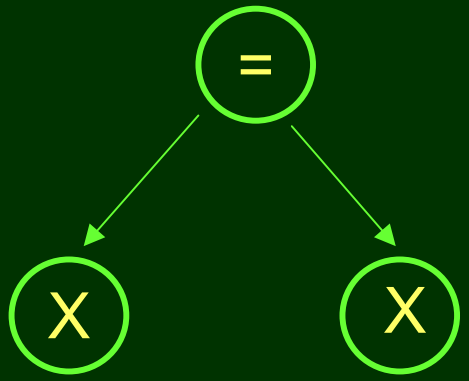
(C) jeśli t_1 jest zmienną, to podstaw $\{t_2/t_1\}$ do wzorca, zbuduj połączenie φ z $\{t_2/t_1\}$ i wywołaj procedurę „match” z tymi wartościami,

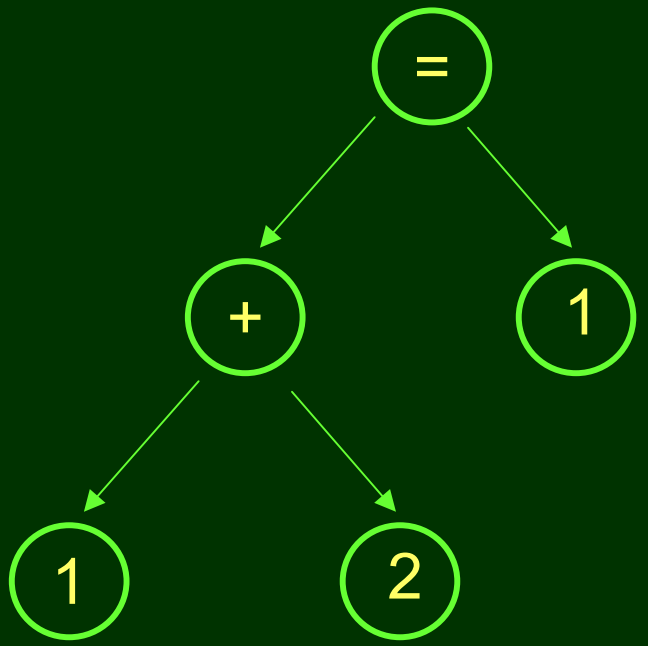
(D) jeśli t_1 nie jest zmienną, to koniec - wyrażenie nie pasuje do wzorca.

Przykład

Czy wyrażenie 1+2=1 pasuje do wzorca X=X ?

Budujemy drzewa rozbioru składniowego obu wyrażeń.

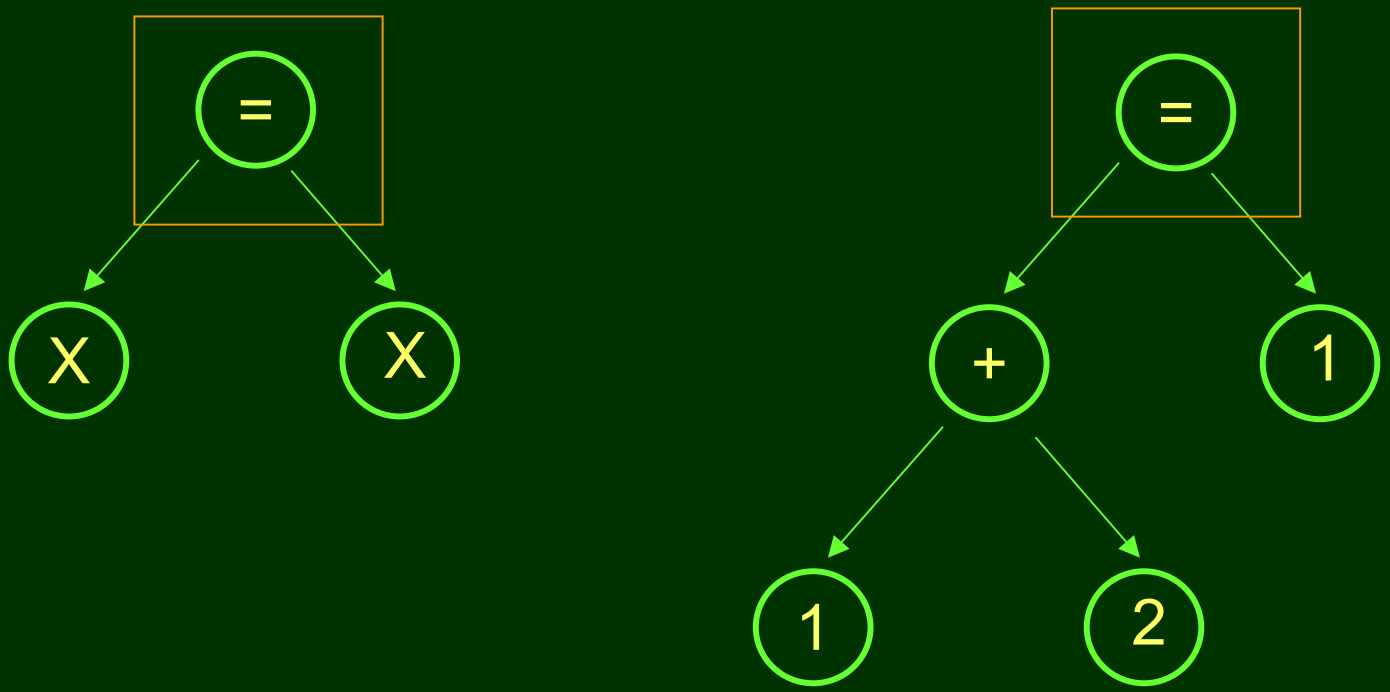




Przykład

Czy wyrażenie 1+2=1 pasuje do wzorca X=X ?

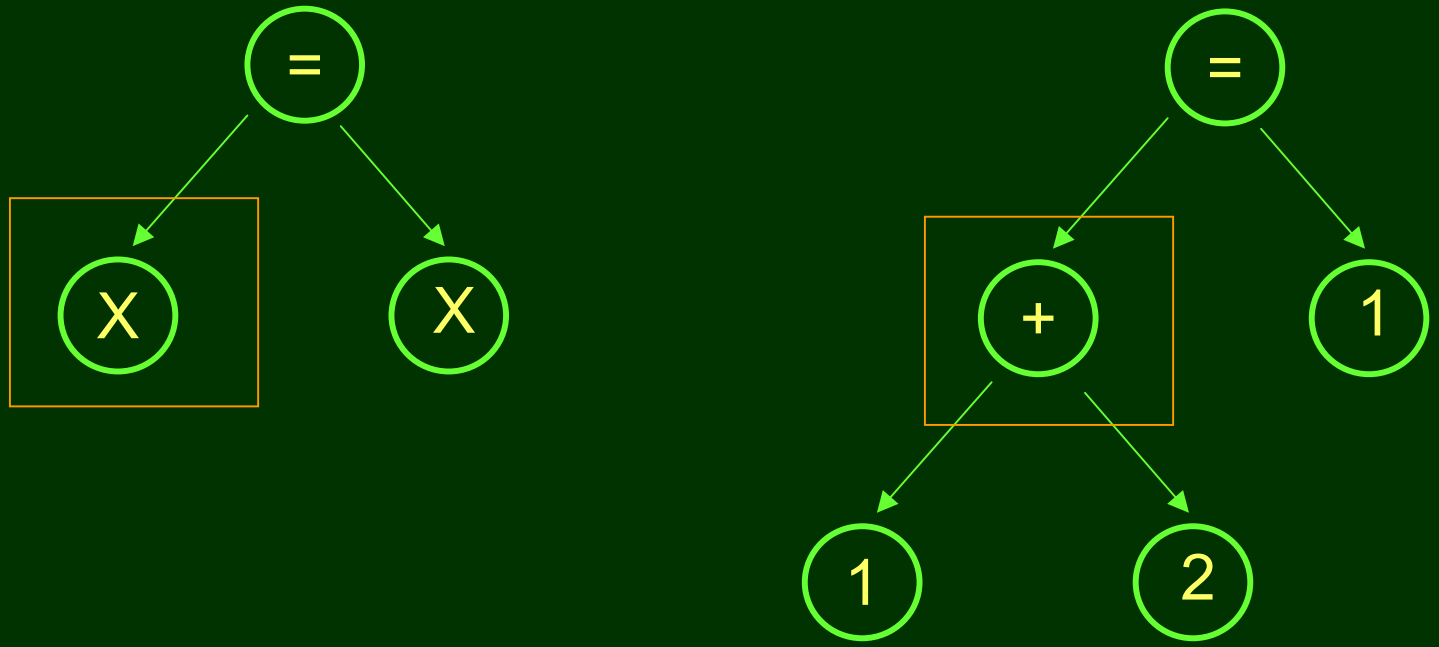
Przeszukujemy jednocześnie oba drzewa w tym samym porządku



Przykład

Czy wyrażenie 1+2=1 pasuje do wzorca X=X ?

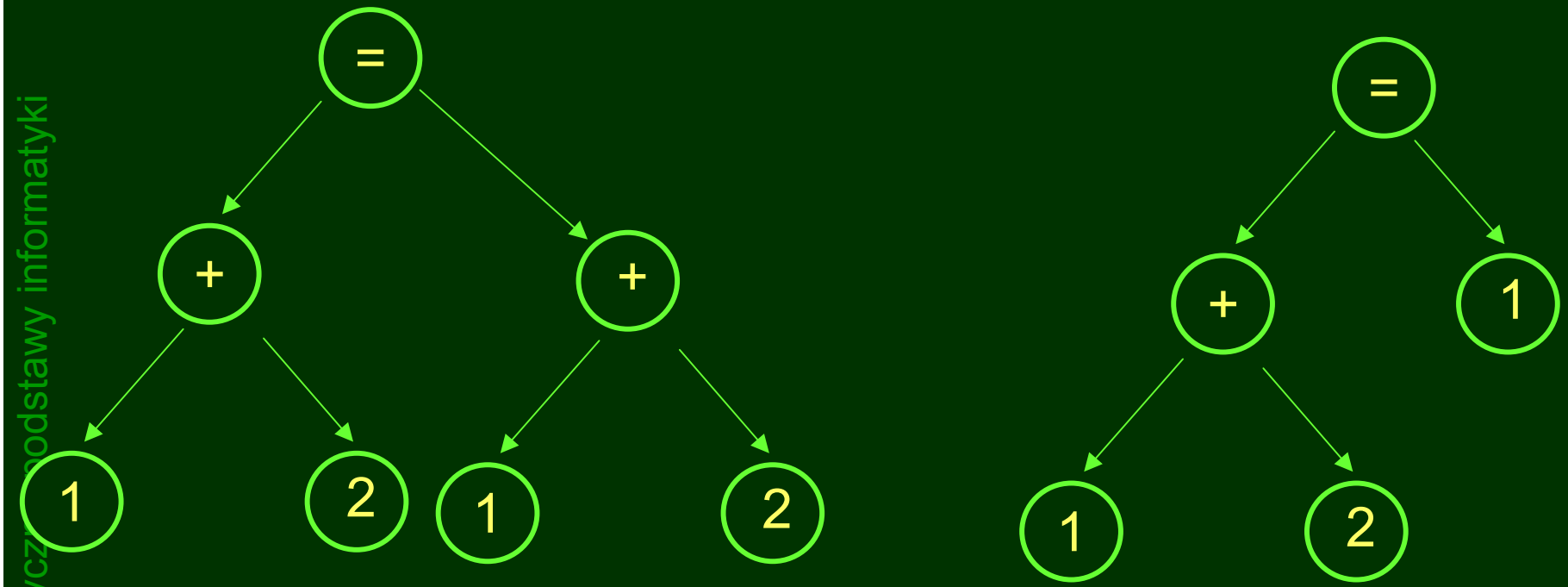
Znajdujemy parę wierzchołków o różnych etykietach; wierzchołki te stanowią korzenie procedur będących parą niezgodną



Przykład

Czy wyrażenie 1+2=1 pasuje do wzorca X=X ?

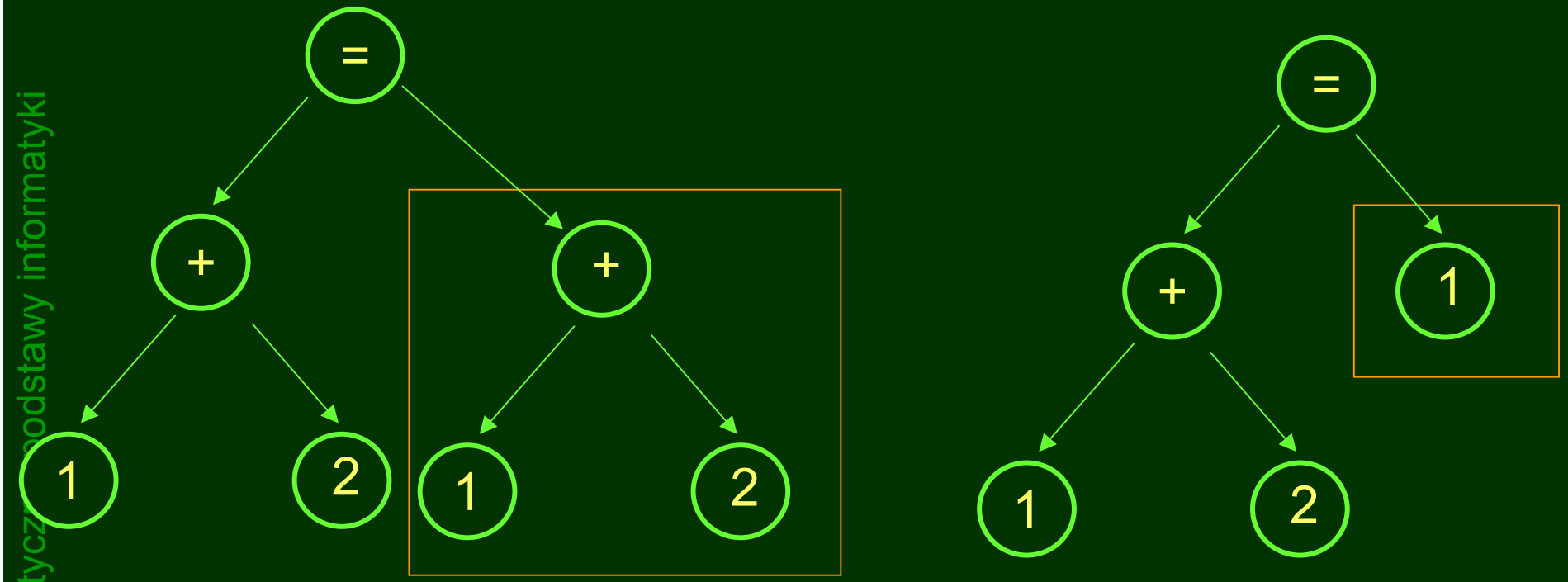
Dokonujemy podstawienia {1+2/X} do wzorca otrzymując φ = {1+2/X}



Przykład

Czy wyrażenie 1+2=1 pasuje do wzorca X=X ?

Następną parą niezgodną jest ⟨1+2, 1⟩, zgodnie z punktem D procedury „match” nie można dopasować 1+2=1 do wzorca X=X.



Składanie podstawień

Podstawienia pokrywają się, jeżeli:

(1) zawierają tę samą zmienną, na którą następuje podstawienie, np. $\varphi_1 = \{a/X\}$ i $\varphi_2 = \{b/X\}$.

(2) zawierają tę samą zmienną, przy czym, jedno zastępuje tę zmienną, a w drugim występuje ona w termie po lewej stronie znaku zastąpienia, np. $\varphi_1 = \{g(Y)/X\}$ i $\varphi_2 = \{b/Y\}$.

Jeżeli podstawienia nie pokrywają się, to złożeniem podstawień jest suma mnogościowa tych podstawień.

Procedura „combine” składania podstawień

Dane są podstawienia θ i φ

(A) zastąp każdą parę s/X w podstawieniu θ przez sφ/X (przez sφ oznaczamy zastosowanie podstawień φ do termu s), uzyskasz w ten sposób podstawienie θ’;

(B) usuń z podstawienia φ każdą t/Y taką, że podstawienie θ’ zawiera parę s/Y, uzyskasz podstawienie φ’;

(C) utwórz sumę mnogościową podstawień θ’ i φ’.

Procedura „combine” ma algebraiczną własność łączności, ale nie jest przemienna.

Przykłady

$\theta = \{a/X, g(a)/Y\}$ $\varphi = \{a/Z\}$ $\theta' = \theta$ i $\varphi' = \varphi$

$\psi = \{a/X, g(a)/Y, a/Z\}$

$\theta = \{g(Y)/X, b/Z\}$ $\varphi = \{a/Y\}$

$\theta' = \{g(Y)\{a/Y\}/X, b/Z\} = \{g(a)/X, b/Z\}$

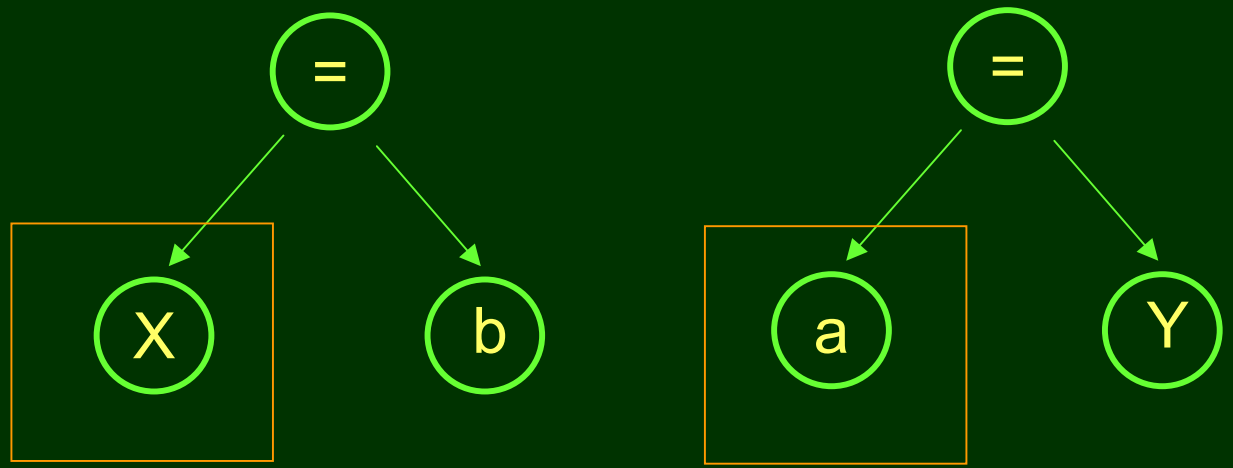
$\varphi' = \{a/Y\}$

$\psi = \{g(a)/X, b/Z, a/Y\}$

Unifikacja

Niech wyrażenie X = b będzie wzorcem.

Czy można dopasować do niego wyrażenie a = Y ?

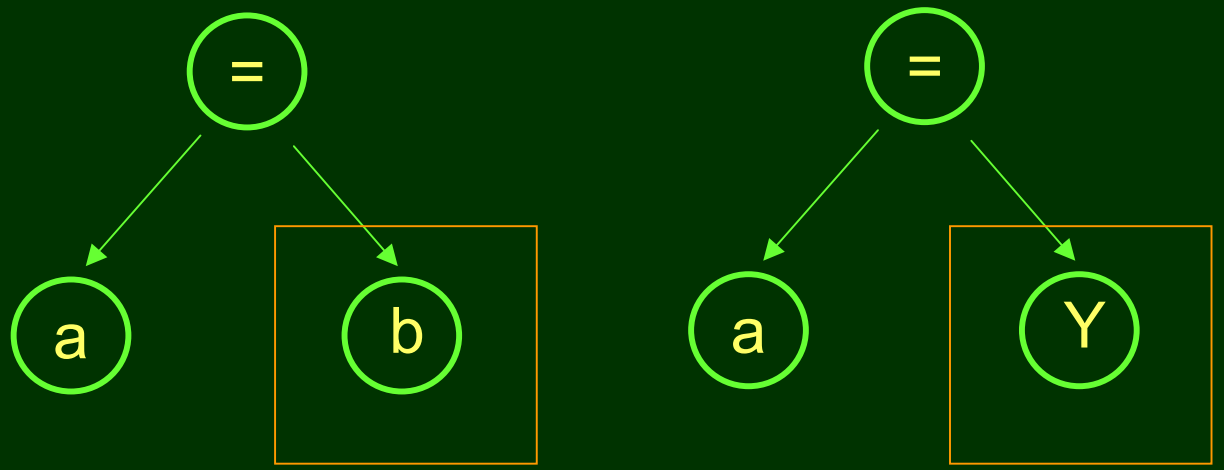


$\varphi = \{a/X\}$

Unifikacja

Niech wyrażenie X = b będzie wzorcem.

Czy można dopasować do niego wyrażenie a = Y ?



b i Y nie można dopasować

Unifikacja

Jeżeli dopuścimy możliwość podstawiania termów za zmienne w obu wyrażeniach (A nie tylko we wzorcu), to można doprowadzić do ujednolicenia (uzgodnienia) powyższych wyrażeń.

Unifikacja jest algorytmem wyznaczającym podstawienia potrzebne do uzgodnienia dwóch wyrażeń rachunku predykatów.

Zasady unifikacji

- wszystkie zmienne muszą występować pod kwantyfikatorem ogólnym,
- zmienne występujące pod kwantyfikatorem szczegółowym:
  - usuwamy kwantyfikator i zastępujemy X stałą Skolema,

    np. $\exists X\ (X^2 + 2X + 1 = 0)$ $a^2 + 2a + 1 = 0$
  - jeżeli kwantyfikator jest zdominowany, to X zastępujemy funkcją Skolema,

    np. $\forall A\ \forall B\ \forall C\ \exists X\ AX^2 + BX + C = 0$

    $AX(A, B, C)^2 + BX(A, B, C) + C = 0$
- nie można za zmienną podstawić wyrażenia zawierającego tę zmienną,
- podstawienie należy wykonać we wszystkich wystąpieniach zmiennej.

Algorytm „unify”

φ – przed wywołaniem procedury jest zbiorem pustym

(A) jeżeli wyrażenia są identyczne, to koniec - wynikiem jest uzyskane podstawienie φ,

(B) niech $\langle t_1, t_2 \rangle$ będzie parą niezgodną,

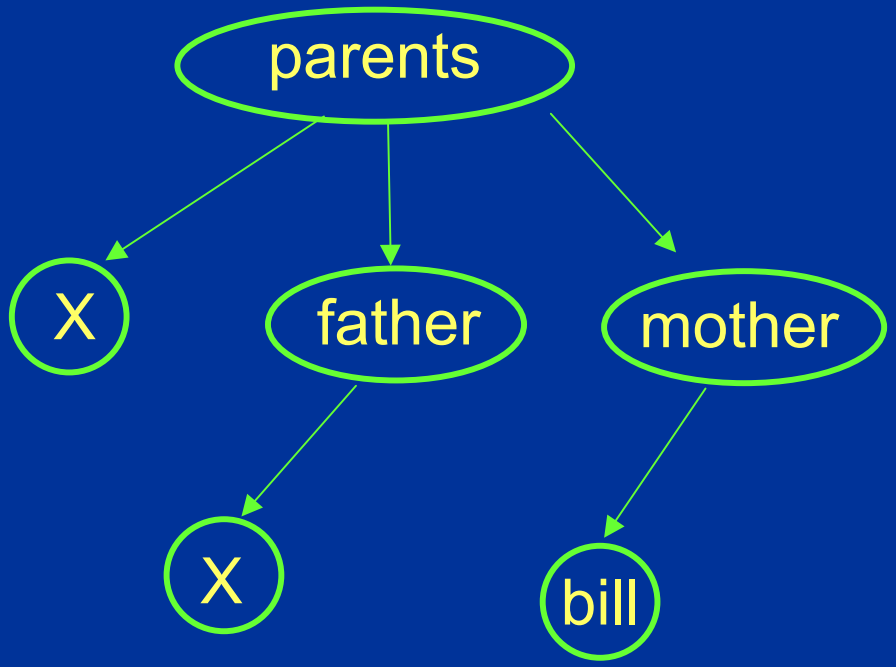
(C) jeśli t_1 jest zmienną, to podstaw $\{t_2/t_1\}$ do obu wyrażeń, złóż φ z $\{t_2/t_1\}$ i wywołaj procedurę „unify” z tymi wartościami,

(D) jeśli t_2 jest zmienną, to podstaw $\{t_1/t_2\}$ do obu wyrażeń, złóż φ z $\{t_1/t_2\}$ i wywołaj procedurę „unify” z tymi wartościami,

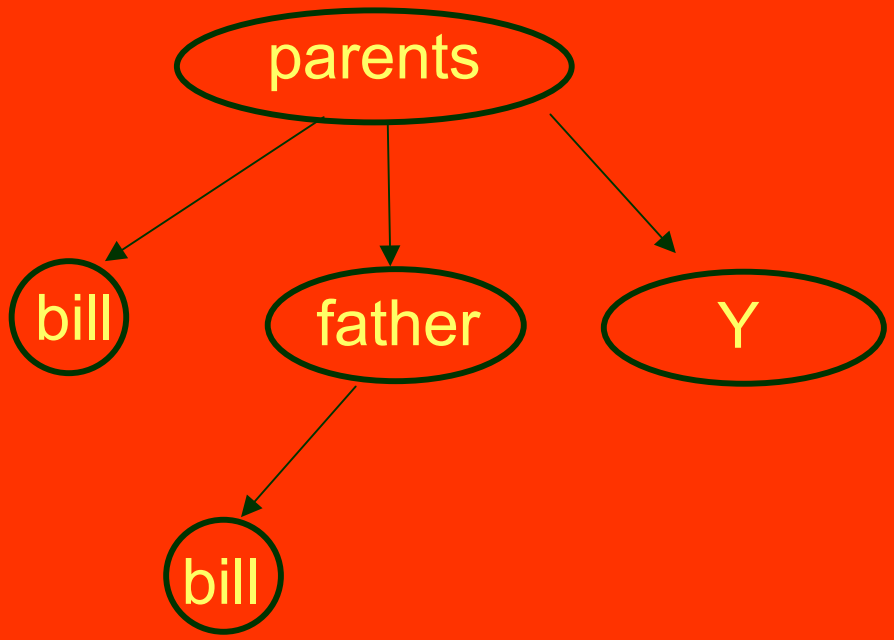
(E) jeśli nie zachodzi ani (C) ani (D), to koniec - wyrażeń nie można uzgodnić.

Unifikacja

parents X(father X)(mother bill)



parents bill(father bill)Y



Unifikacja

parents X(father X)(mother bill)

parents bill(father bill)Y

Unifikacja

parents X(father X)(mother bill)

bill(father bill)Y

{X/bill}

Unifikacja

X(father X)(mother bill)

bill(father bill)Y

{X/bill}

Unifikacja

(father bill)(mother bill)

(father bill)Y

{X/bill}

Unifikacja

bill (mother bill)

bill Y

{X/bill}

Unifikacja

(mother bill)

Y

{X/bill, Y/(mother bill)}

Unifikacja

parents bill(father bill)(mother bill)

parents bill(father bill)(mother bill)

{X/bill, Y/(mother bill)}

Złożenie unifikatorów

Jeżeli *s* i *s'* są zbiorami podstawień, to złożenie *s's* otrzymuje się stosując podstawienia *s* do elementów *s'* i dodając wynik do *s*.

$s = \{Y/X, Z/W\}$, $s' = \{Z/V\}$

Podstawienia *s* do elementów *s'*: $\{W/V\}$

Dodanie wyniku do *s*: $s's = \{Y/X, Z/W, W/V\}$

Złożenie unifikatorów jest łączne, ale nie jest przemienne!

$s = \{Y/X, Z/W\}$, $s' = \{Z/V\}$

Podstawienia *s'* do elementów *s*: $\{Y/X, V/W\}$

Dodanie wyniku do *s'*: $ss' = \{Z/V, Y/X, V/W\}$

Najogólniejszy unifikator

Jeżeli *g* jest najogólniejszym unifikatorem wyrażeń *E*, to dla dowolnego *s* będącego unifikatorem tego zbioru wyrażeń, istnieje unifikator *s'*, taki że *Es* = *Egs'*, gdzie gs' oznacza złożenie unifikatorów *g* i *s'*.

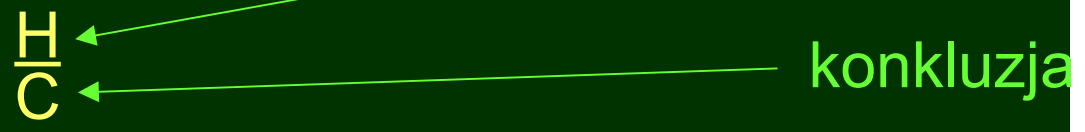
Procedura „unify" znajduje najogólniejszy unifikator pary wyrażeń.

Interpretacja języka FOL

- Każdej zmiennej ze zbioru Z można przypisać wartość z pewnego zbioru, każdemu n-argumentowemu symbolowi funkcyjnemu przypisujemy n-argumentową funkcję, a n-argumentowemu symbolowi predykatu n-arną relację.
- Funkcja interpretująca termy i formuły (funkcja semantyki) FOL wyznacza ich wartości przy założeniu pewnego systemu interpretacji (np. systemu relacyjnego).
- Formuła jest spełnialna, jeżeli istnieje taka interpretacja i takie przypisanie zmiennym wartości, dla których formuła uzyskuje wartość „prawda”.
- Formuła jest prawdziwa (jest tautologią), jeśli jest spełnialna dla każdej interpretacji.

Interpretacja języka FOL

- O danej interpretacji mówimy, że jest modelem formuły jeśli dla każdego wartościowania zmiennych przy tej interpretacji formuła przyjmuje wartość „prawda”.
- Reguła wnioskowania postaci:



jest poprawna, gdy każda interpretacja formuł H i C, która jest modelem H jest również modelem C; mówimy, że C jest logiczną konsekwencją H, co zapisujemy $H \models C$,

- Dowodem nazywamy sekwencję formuł, z których każda jest albo aksjomatem albo została wywiedziona z wcześniejszych formuł po zastosowaniu reguły wnioskowania.

Interpretacja języka FOL

- Jeśli formuła α jest dowiedlna (posiada dowód), to nazywamy ją twierdzeniem (co zapisujemy $\vdash \alpha$).
- Z twierdzenia o pełności dla FOL (K. Gödel) wynika, że formuła jest dowiedlna wtedy i tylko wtedy, gdy jest prawdziwa, dzięki czemu problem stwierdzenia prawdziwości formuły sprowadza się do znalezienia jej dowodu, którego konstrukcja jest operacją czysto składniową (tekstową), a więc stosunkowo łatwo poddającą się automatyzacji.
- Dowodzenie twierdzeń w FOL ogranicza własność częściowej rozstrzygalności tej logiki, tzn. nie istnieje automatyczna procedura rozstrzygająca, czy dana formuła jest twierdzeniem, czy też nie, jednakże istnieje procedura znajdująca dowód, jeśli formuła jest dowiedlna.

Interpretacja języka FOL

- Jedną z metod automatycznego dowodzenia twierdzeń w FOL, przy wykorzystaniu metody „nie wprost” jest odkryta w roku 1965 przez J. B. Robinson tzw. zasada rezolucji, oparta na jednej regule wnioskowania zwanej regułą rezolucji.
- Dowód metodą rezolucji wymaga sprowadzenia wszystkich formuł do „normalnej postaci klauzulowej”.

Określenia

- *Literał dodatni* – formula atomowa
- *Literał ujemny* – negacja formuly atomowej
- α *oraz* $\neg\alpha$ – klauzule komplementarne
- *Klauzula Horna* – dysjunkcja (alternatywa) literałów, zawierajaca co najwyżej jeden literał dodatni

Formuły w postaci normalnej

- Spójniki $\Rightarrow$, $\Leftrightarrow$ zamieniamy na równoważne $\wedge$, $\vee$, $\neg$;
- Wszystkie kwantyfikatory umieszcza się na początku formuły, a następnie usuwa (skolemizacja)

Przykład:

$\exists Z \forall X\,(R(Z) \wedge P(X)) \Rightarrow (\exists Y\, Q(X,Y))$

$\forall Z\, \exists X\, \neg R(Z) \vee \neg P(X) \vee \exists Y\, Q(X,Y)$

$\exists Y\, \forall Z\, \exists X\, (\neg R(Z) \vee \neg P(X) \vee Q(X,Y))$

Formuły w postaci koniunkcyjnej

- Formuła jest koniunkcją alternatyw literałów
  - etap 1: $\neg \neg A \Rightarrow A$ $\quad \neg (A \vee B) \Rightarrow \neg A \wedge \neg B$
    $\neg (A \wedge B) \Rightarrow \neg A \vee \neg B$
  - etap 2: $A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge (A \vee C)$
    $(B \wedge C) \vee A \Rightarrow (B \vee A) \wedge (C \vee A)$

Przykład:

$\neg (P(X) \vee Q(X,Y)) \vee R(Z)$

$(\neg P(X) \wedge \neg Q(X,Y)) \vee R(Z)$

$\neg (P(X) \vee R(Z)) \wedge (\neg Q(X,Y) \vee R(Z))$

Formuły w postaci klauzulowej

- Formułę w postaci koniunkcyjnej zamieniamy na zbiór formuł (z przemianowaniem zmiennych)

Przykład:

$\neg (P(X) \vee R(Z)) \wedge (\neg Q(X,Y) \vee R(Z))$

$\{\neg (P(X) \vee R(Z)), (\neg Q(X_1,Y) \vee R(Z_1))$

Dowód metodą nie wprost

- Założenie o niesprzeczności teorii
- Zatem dodanie zdania, które nie jest prawdziwe powoduje powstanie sprzeczności w rozważanym zbiorze zdań
- Szukamy tej sprzeczności
- Co będzie jeżeli nie ma sprzeczności?

Rezolucja binarna

$$\frac{C' \vee P', C'' \vee P''}{(C' \vee C'')\varphi}$$

P’ i P” są literałami komplementarnymi po zastosowaniu unifikacji, przy wykorzystaniu podstawienia φ.

$$\frac{\alpha \vee \beta, \neg\alpha \vee \gamma}{\beta \vee \gamma}$$

Klauzule w postaci Kowalskiego

- Klauzulę postaci

  $\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_m \vee Q_1 \vee Q_2 \vee \ldots \vee Q_n$

  można przepisać do:

  $P_1 \wedge P_2 \wedge \ldots \wedge P_m \Rightarrow Q_1 \vee Q_2 \vee \ldots \vee Q_n$

  zwanej postacią Kowalskiego.
- Reguła rezolucji przyjmuje postać:

$$\frac{\left.\begin{array}{l} H' \Rightarrow C' \vee P'_1 \vee P'_2 \vee \ldots \vee P'_m\,, \\ H' \wedge P''_1 \wedge P''_2 \wedge \ldots \wedge P''_n \Rightarrow C'' \end{array}\right\}\ \text{przesłanki}}{(H' \wedge H'' \Rightarrow C' \vee C'')\varphi}\ \text{rezolwenta}$$

Teoria przemijania

A1. $die(X) \vee \neg man(X)$

A2. $man(sokrates)$

Tw. o Sokratesie: $die(sokrates)$

Dowód: 1. zaprzeczenie tezy:

Z1. $\neg\ die(sokrates)$

2. na podstawie reguły podstawiania: (X/sokrates)

A1'. $die(sokrates) \vee \neg man(sokrates)$

3. rezolucja Z1 oraz A1'

Z2. $\neg man(sokrates)$

4. rezolucja A2 i Z2 powoduje wygenerowanie klauzuli pustej, co kończy dowód.

Dowód metodą rezolucji

1. Przekształć przesłanki lub aksjomaty w formę klauzul.
2. Dodaj do zbioru aksjomatów zaprzeczenie twierdzenia, które ma być udowodnione.
3. Wygeneruj nowe klauzule wynikające z tego zbioru.
4. Znajdź sprzeczność generując pustą klauzulę.
5. Warunki użyte do wygenerowania pustej klauzuli są tymi, w których zaprzeczenie celu jest prawdziwe.

Implementacja dowodzenia metodą rezolucji

- Wybór klauzuli początkowej
  - zbiór minimalnie niespełnialny,
  - wybór celu (jeśli jest twierdzeniem).
- Wybór literału aktywnego
  - jest dowolny (wszystkie literały muszą być usunięte),
  - sztywna zasada (pierwszy od lewej) nie wpływa na zupełność strategii liniowej, drzewo poszukiwań znacznie się upraszcza.

Implementacja dowodzenia metodą rezolucji

- Wybór klauzuli
  - przeszukiwanie w głąb (DFS) - nie jest zupełna dowód nie musi być najkrótszy,
  - przeszukiwanie wszerz (BFS) - zupełna, ale nieefektywna,
  - przeszukiwanie heurystyczne (np. długość klauzuli),
  - przeszukiwanie w głąb z ograniczoną głębokością (CB-DFS: consecutively bounded DFS).

Rezolucja w PROLOGU

- Zbiory klauzul Horna można traktować jak instrukcje języka programowania, a program dowodzący, bazujący na liniowej, uporządkowanej rezolucji ze strategią DFS - jako interpreter tego języka.
- Wiązania zmiennych, powstające w trakcie unifikacji można traktować jak reprezentację wyników obliczeń.
- Idea „programowania w logice” pochodzi od R.Kowalskiego. Rozwijali ją A.Colmerauer, L.Pereira, D.Warren i inni.

Cechy rezolucji prologowej:

- realizacja strategii DFS za pomocą nawracania, w sytuacji, gdy dalszy wywód nie jest możliwy (backtracking on failure),
- unifikacja bez testu wystąpienia (occurence check) - przyspieszenie obliczeń, nieskończone drzewa jako struktury danych,
- sterowanie przebiegiem obliczeń: odcięcie (cut) i wymuszenie nawrotu (fail)

Przykład

$\rightarrow$ append(nil, List, List)

append(Tail, List, Res) $\rightarrow$

append(cons(Head, Tail), List, cons(Head, Res)

$\rightarrow$ append([], List, List).

append([H/T], List, [H/Res]) :- append(T, List, Res).

append([one, thing], [and, another], Result)

{cons(one, Result'/Result}

append([thing], [and, another], Result')

{cons(thing, Result"/Result'}

append(nil, [and, another], Result")

{cons([and, another]/Result"}

→