sowa WERSJA 4, pytania <= 2021

poprawne odpowiedzi są **pogrubione**

1. Mając na uwadze fakt, iż negacja w języku Prolog realizowana jest przez niepowodzenie należy dla faktów q(a,b). q(a,c). oraz reguły s(X,Y) :- not(X=Y), q(Z,X), q(Z,Y). wybrać odpowiedź będącą rezultatem zapytania ?- s(P,R). :
   - **False**
   - P=_G327 R=_G417
   - P=b R=c; P=c R=b; False
   - P=a R=b; P=b R=c; P=a R=c; False
2. Elementy programu prologowego, które decydują o jego interpretacji proceduralnej to:
   - **kolejność reguł w programie**
   - sumaryczna liczba odcięć w regułach
   - **kolejność warunków w regułach**
   - **umiejscowienie odcięć w regułach**
   - sumaryczna liczba reguł w programie
3. Weryfikacja rodzaju termu odbywa się w języku Prolog za pomocą następujących metapredykatów systemowych:
   - name
   - ~~functor~~
   - **atom**
   - ~~setof~~
   - **var**
   - arg
   - **atomic**
4. Do rodziny predykatów, umożliwiających automatyczną generację (w postaci listy) wszystkich rozwiązań celu, uzyskiwanych w drodze nawrotów zaliczamy następujące metapredykaty systemowe:
   - repeat
   - **bagof**
   - forall
   - **findall**

- **setof**
- call
- assert

5. Predykaty użytkownika mogą być argumentami wywołania następujących metapredykatów systemowych:
   - repeat
   - **bagof**
   - **setof**
   - **findall**
   - **assert**
   - **not**
   - **call**

6. Które z poniższych zapytań zakończą się spełnieniem celu :
   - ?- X=Z, Y=Z, X\==Y.
   - **?- X=Z, Y=Z, X==Y.**
   - ?- X=a, Y=a, X\==Y.
   - **?- X=a, Y=a, X==Y.**
   - **?- _\==_.**
   - ?- _==_
   - **?- X=f(_), Y=f(_), X\==Y**.
   - ~~?- X=f(_), Y=f(_), X==Y.~~
   - ?**- f(a,_)\==f(a,_)**
   - ?- f(a,_)==f(a,_)

7. Nagłówek w klauzuli prologowej w sensie formalnym:
   - może zawierać negację predykatu
   - **nie może zawierać negacji predykatu**
   - może zawierać operatory koniunkcji lub dysjunkcji
   - nie może zawierać operatorów koniunkcji i dysjunkcji
     (nie może ani tego ani koniunkcji, ani dysjunkcji - niejasno sformułowana odpowiedź)
   - **nie może zawierać operatorów koniunkcji lub dysjunkcji**
   - **może zawierać predykatu o arności 0**
   - nie może zawierać predykatu o arności 0
   - **może być pusty**
   - nie może być pusty

8. Odcięcie:

   – **uniemożliwia spełnienie celów poprzedzających je w części warunkowej klauzuli w inny, alternatywny sposób**

   – **uniemożliwia analizę wszystkich pozostałych klauzul z tym samym nagłówkiem, które znajdują się w programie poniżej klauzuli, w której odcięcie wystąpiło**

   – powoduje natychmiastowe spełnienie celu, który został dopasowany do nagłówka klauzuli je zawierającej

   – powoduje, że występujące za nim cele w części warunkowej klauzuli nie są analizowane

9. Przetwarzanie składowych złożonych struktur danych (reprezentowanych w języku Prolog za pomocą zagnieżdżonych termów) realizowane jest z wykorzystaniem następujących operatorów i metapredykatów systemowych:

   – call

   – atomic

   – assert

   – **=..**

   – var

   – **arg**

   – setof

   – **functor**

10. Do ewaluacji wartości wyrażenia będącego termem złożonym z funktorem arytmetycznym dochodzi, kiedy użyjemy operatora:

   – **is**

   – **<**

   – **=:=**

   – **>**

   – **=\=**

   – **=**

   – **=..**

   – **==**

   – **>=**

   – **=<**

11. Mając na uwadze fakt iż negacja w języku Prolog realizowana jest przez niepowodzenie należy dla reguły postaci: s(X,Y) :- not(X=Y), r(Z, X), r(Z, Y). oraz dwóch faktów: r(a, m). r(a, n). Wybrać odpowiedzi będące rezultatem zapytania: ?-s(P,R).

   a) P=n R=n; False
   b) P=n R=n; P=n, R=m; False
   c) False
   d) P=n R=m; P=m R=n; P=n R=m; P=n R=n; False

14. Jaki będzie rezultat wykonania operacji uzgodnienia

```
?- p(a,X,f(g(X))) = p(Z,f(Z),f(W)).
```

a) `X = f(a), Z = a, W = X.`
b) `X = f(a), Z = a, W = g(f(a)).`
c) `False.`
d) `X = Z, Z = a, W = g(f).`

17. Jaki będzie wynik wykonania zapytania:

```
?- [a,[b,c]]=..[X,_,Z].
```

a) `X='.' Z=[b,c]`
b) `X='|' Z=[[b,c]]`
c) `X='.' Z=[[b,c]]`
d) `X='|' Z=[b,c]`

18. Do grupy predykatów dekompozycji wyrażeń języka Prolog zaliczamy następujące metapredykaty systemowe:

a) `call`
b) `assert`
c) `arg`
d) `findall`

19. Jaki będzie rezultat wykonania operacji uzgadniania:

```
?- n(k,Y,g(h(Y)),h(Y))=n(W,p(W),g(X),h(p(W))).
```

a) `X=h(p(k))  W=k  Y=X`
b) `False`
c) `X=h(k)  W=k  Y=h(p(k))`
d) `X=h(p(k))  W=k  Y=p(k)`

8. Jaki będzie wynik wykonania zapytania `?- [a,[b,c]] =.. [_, X, Y]`

a) `X=[]  Y=[a,[b,c]]`
b) `X=a  Y=[[b,c]]`
c) `X=a  Y=[b,c]`
d) `X=[a]  Y=[b,c]`

Functor w zapytaniach

1. Jaki będzie wynik zapytania: ?- functor(X,'.',2), arg(1,X,a), arg(2,X,[b]) :
   - X=[a,b,[]]
   - X=[a,[b]]
   - **X=[a,b]**
   - X=[a|b]
2. Jaki będzie wynik wykonania zapytania ?- functor([A|[B|[C]]], F, N). :
   - **F='.' N=2**
   - F='|' N=3
   - F="," N=2
   - F='.' N=3
3. Jaki będzie wynik wykonania zapytania ?- functor(f(m,n,n),F, N). :
   - F='.' N=2

- **F='f' N=3**
- False
- F=',' N=3

Zapytania z arg

1. Jaki będzie wynik wykonania zapytania ?- arg(2,[a,b,c,d],K).:
   - K=[b]
   - False
   - **K=[b,c,d]**
   - K=b
2. Jaki będzie wynik wykonania zapytania ?- arg(3,[a,b,c,d],K).:
   - K=[c,d]
   - **False**
   - K=[c]
   - K=c
3. Jaki będzie wynik wykonania zapytania ?- arg(2,[[a],[b],[c],[d]],K).:
   - False
   - K=[[b]]
   - **K=[[b],[c],[d]]**
   - K=[b]
4. Jaki będzie wynik wykonania zapytania ?- arg(3,[[a],[b],[c],[d]],K).:
   - **False**
   - K=c
   - K=[c]
   - K=[[c],[d]]

Zapytania z is

1. Jaki będzie wynik wykonania zapytania ?- N is 6, K=4, K is N-2. :
   - **N=6 K=4**
   - N=4 K=2
   - N=4 K=4
   - False

2. Jaki będzie wynik wykonania zapytania ?- N is 7, K=5, K is N-2.:
   - N=5 K=5
   - **N=7 K=5**
   - N=7 K=3
   - False
3. Jaki będzie wynik zapytania ?- N is 6, K=5, K is N-2.:
   - N=6 K=3
   - **False**
   - N=6 K=4
   - N=6 K=5
4. Jaki będzie wynik zapytania ?- K=4, N=K, N is K-2. :
   - K=2 N=2
   - K=4 N=2
   - **False**
   - K=4 N=4
5. Jaki będzie wynik wykonania zapytania ?- N is 7, K=4, K is N-2. :
   - N=7 K=3
   - N=7 K=4
   - N=7 K=5
   - **False**

Interpretacje deklaratywne

1. Która z interpretacji deklaratywnych definicji klauzuli `p:-!,a,b. p:-!,c,d. p:-e.` jest kompletna i poprawna:
   - **p<=>(a AND b)**
   - p<=>(a AND b) OR (c AND d) OR e
   - p<=>(a AND b) OR (¬a AND ¬c AND e)
   - p<=>(¬a AND b) OR (¬c AND d) OR e
2. Która z interpretacji deklaratywnych klauzuli `p:-a,!,b. p:-c,!,d. p:-e.` jest kompletna i poprawna:
   - p<=>(a and ¬b) OR (c AND ¬d) OR e
   - p<=>(¬a AND b) OR (¬c aND d) OR e
   - **p<=>(a AND b) OR (¬a AND c AND d) OR (¬a AND ¬c AND e)**
   - p<=>(¬a AND ¬c AND e)

3. Która z interpretacji deklaratywnych klauzuli `p:-!,a,!,b,c. p:-d.` jest kompletna i poprawna:
   - p<=>(¬a AND ¬b AND c) OR d
   - p<=>(a AND b AND c) OR (¬a AND d) OR (¬b AND d)
   - p<=>(a AND b AND c) OR (¬a AND ¬b AND d)
   - **p<=>(a ANd b AND c)**
4. Która z interpretacji deklaratywnych definicji klauzuli `p:-a,!,b. p:-!,c,d. p:-e.`
   - **p<=>(a AND b) OR (¬a AND c AND d)**
   - p<=>(¬a AND e)
   - p<=>(a AND ¬b) OR (¬c AND d) OR e
   - p<=>(¬a AND b) OR (c AND d) OR e
5. Która z interpretacji deklaratywnych klauzuli `p:-a,!,b,!,c. p:-d.` jest kompletna i poprawna:
   - **p<=>(a AND b AND c) OR (¬a AND d)**
   - p<=>(a AND b AND c) OR (¬a AND d) OR (¬b AND d)
   - p<=>(a AND ¬b AND ¬c) OR d
   - p<=>(a AND b AND c) OR (¬a AND ¬b AND d)
6. Która z interpretacji deklaratywnych klauzuli `p:-!,a,b. p:-c,!,d. p:-e.` jest kompletna i poprawna: (?)
   - p<=>(a AND b) OR (¬a AND c AND e)
   - p<=>(¬a AND b) OR (c ANd ¬d) OR e
   - **p<=>(a AND b)**
   - p<=>¬(a AND b) OR (c AND ¬d) OR e
7. Która z interpretacji deklaratywnych definicji klauzuli `p:-!,a,b,!,c. p:-d.` jest kompletna i poprawna:
   - p<=>(¬a AND b AND ¬c) OR d
   - p<=>(a AND b AND c) OR (¬a AND d) OR (¬c AND d)
   - **p<=>(a AND b AND c)**
   - p<=>(a AND b AND c) OR (¬a AND b AND ¬d)

Operatory

1. Dla definicji operatorów op(100,xfy,#). oraz op(100,fy,@). wyrażenie a # @ b # c jest:
   - **równoważne wyrażeniu: a # (@ b # c)**
   - niepoprawne
   - równoważne wyrażeniu: a # ((@ b) # c)
   - **równoważne wyrażeniu: a # @ (b # c)**

2. Dla definicji operatorów op(100, xfy, ^). oraz op(50, fy, ~). wyrażenie a ^ ~ b ^ c jest:
   - niepoprawne
   - równoważne wyrażeniu: a ^ ~ (b ^ c)
   - **równoważne wyrażeniu: a ^ ((~ b) ^ c)**
   - **równoważne wyrażeniu: a ^ (~ b ^ c)**
3. Dla definicji operatorów op(100, xfy, #). oraz op(50, xf, @). wyrażenie a # b @ # c jest"
   - niepoprawne
   - równoważne wyrażeniu: (a # b @) # c)
   - **równoważne wyrażeniu: a # ((b @) # c)**
   - równoważne wyrażeniu: (a # b) @ # c)
4. Dla definicji operatorów op(100, xfy, ^). oraz op(100, fy, ~). wyrażenie a ^ ~ b ^ c jest:
   - niepoprawne
   - **równoważne wyrażeniu: a ^ ~ (b ^ c)**
   - równoważne wyrażeniu: a ^ ((~ b) ^ c)
   - **równoważne wyrażeniu: a ^ (~ b ^ c)**
5. Dla definicji operatorów op(100,xfx,^). oraz op(50,xf,~). wyrażenie a ^ b ~ ^ c jest:
   a) równoważne wyrażeniu: (a ^ b) ~ ^ c
   b) równoważne wyrażeniu: a ^ ((b ~) ^ c)
   c) równoważne wyrażeniu: (a ^ b ~) ^ c
   d) niepoprawne
6. Dla definicji operatorów `op(100,xfx,^).` oraz `op(100,fy,~).` Wyrażenie `a ^ b ~ b ^ c` jest:
   a) równoważne wyrażeniu: `a^ (~b ^ c)`
   b) równoważne wyrażeniu: `a^ ((~ b) ^ c)`
   c) równoważne wyrażeniu: `a^ ~ (b ^ c)`
   d) niepoprawne
7. Dla definicji operatorów `op(100, xfy, #).` oraz `op(55, xf, ~).` wyrażenie `a # b ~ # c` jest:
   a) równoważne wyrażeniu `(a # b) ~ # c`
   b) równoważne wyrażeniu `(a # b ~) # c`
   c) **równoważne wyrażeniu `a # (b ~) # c`**
   d) niepoprawne

member/append-var/nonvar

1. Jaki będą wszystkie możliwe wyniki wykonania niedeterministycznego zapytania: ?- member(X,[a,A,b,B,c,C]), var(X). :
   - **X=A; X=B; X=C; False**
   - X=a; False
   - False
   - X=a; X=b; X=c; False
2. Jakie będą wszystkie możliwe wyniki wykonania niedeterministycznego zapytania: ?- member(X,[a,A,b,B,c,C]), nonvar(X). :

- **X=a; X=b; X=c; False**
- X=A; X=B; X=C; False
- False
- X=a; False

3. Jakie będą wszystkie możliwe wyniki wykonania niedeterministycznego zapytania: ?- append([_|_], [X|_], [A,a,B,b,C,c]), var(X). :
   - X=b; X=c; False
   - **X=B, X=C; False**
   - X=A; X=B; X=C; False
   - X=a; X=b; X=c; False
4. Jakie będą wszystkie możliwe wyniki wykonania niedeterministycznego zapytania: ?- append([_|_], [X|_], [A,a,B,b,C,c]), nonvar(X). :
   - X=b; X=c; False
   - X=B, X=C; False
   - X=A; X=B; X=C; False
   - **X=a; X=b; X=c; False**
5. Które z poniższych zapytań wybiera dowolną parę w dowolnej kolejności dwóch nie tych samych elementów z listy L:
   - **?- append(P, [X|R],L), append(P,R,Q), append(_,[Y|_],Q).**
   - ?- append(_, [X|R],L), append(_,[Y|_],R).
   - ?- append(_, [X,Y|_],L).
   - ?- append(_, [X|_],L), append(_,[Y|_],L).
6. Które z poniższych zapytań wybiera dowolną parę dwóch zupełnie dowolnych elementów (bez ograniczeń kolejnościowych z listy L:
   - ?- **append(_,[X|_],L), append(_,[Y|_],L).**
   - ?- append(_,[X|R],L), append(_,[Y|_],R).
   - ?- append(_,[X,Y|_],L).
   - ?- append([X],[_|Y],L).
7. Niech abs oznacza funkcję arytmetyczną, zwracającą wartość bezwzględną liczby. Jaki będzie wynik działania zapytania ?- m([**1,-3,8,-5,2,-4**],X). dla definicji predykatów: m(L,X) :- member(X,L), a(X,L). a(X,L) :- L=[]; L=[H|T], abs(H)**=<**abs(X), a(X,T).
   - X = -4
   - X = -5
   - **X = 8**
   - X = 1

8. Niech abs oznacza funkcję arytmetyczną, zwracającą wartość bezwzględną liczby. Jaki będzie wynik działania zapytania ?- m([1,-3,8,-5,2,-4],X). dla definicji predykatów: m(L,X) :- member(X,L), a(X,L). a(X,L) :- L=[]; L=[H|T], abs(H)>=abs(X), a(X,T).

   - X = -4
   - X = -5
   - X = 8
   - **X = 1**

9. Niech abs oznacza funkcję arytmetyczną, zwracającą wartość bezwzględną liczby. Jaki będzie wynik działania zapytania ?- m([-1,3,-8,5,-2,1],X). dla definicji predykatów: m(L,X) :- member(X,L), a(X,L). a(X,L) :- L=[]; L=[H|T], abs(H)=<abs(X), a(X,T).

   - **X = -8**
   - X = -1
   - X = 5
   - X = 1

10. Niech abs oznacza funkcję arytmetyczną, zwracającą wartość bezwzględną liczby. Jaki będzie wynik działania zapytania ?- m([-1,3,-8,5,-2,1],X). dla definicji predykatów: m(L,X) :- member(X,L), a(X,L). a(X,L) :- L=[]; L=[H|T], abs(H)>=abs(X), a(X,T).

   - **X = -1.**

11. Jaki będzie wynik działania zapytania ?- m([-1,3,-8,5,-2,1],X). dla definicji predykatów: m(L,X) :- member(X,L), a(X,L). a(X,L) :- L=[]; L=[H|T], H>=X, a(X,T).

   - **X = -8.**

Retract

1. Mając na uwadze niedeterminizm predykatu retract należy wskazać, które z poniższych odpowiedzi dla celu ?-g(X). są poprawne, jeżeli wcześniej zostały wydane zapytania ?- asserta(g(3)),asserta(g(2)),asserta(g(1)). ?- retract(g(X)). :

   - **X=2; X=3; False**
   - **X=3; False**
   - X=1; X=2; X=3; False
   - **False**

2. Mając na uwadze niedeterminizm predykatu retract należy wskazać, które z poniższych odpowiedzi dla celu ?-g(X). są poprawne, jeżeli wcześniej zostały wydane zapytania ?- assertz(g(3)),assertz(g(2)),assertz(g(1)). ?- retract(g(X)). :

   - X=3; X=2; False
   - X=3; False
   - X=3; X=2; X=1; False
   - **False**
   - **X=2; X=1; False**

- **X=1; False**

3. Mając na uwadze niedeterminizm predykatu retract należy wskazać, które z poniższych odpowiedzi dla celu ?-g(X). są poprawne, jeżeli wcześniej zostały wydane zapytania ?- asserta(g(1)),asserta(g(2)),asserta(g(3)). ?- retract(g(X)). :

   - X=3; X=2; False
   - X=3; False
   - X=3; X=2; X=1; False
   - **False**

4. Mając na uwadze niedeterminizm predykatu retract należy wskazać, które z poniższych odpowiedzi dla celu ?-g(X). są poprawne, jeżeli wcześniej zostały wydane zapytania ?- assertz(g(1)),assertz(g(2)),assertz(g(3)). ?- retract(g(X)). :

   - X=1; X=2; False
   - X=1; False
   - X=1; X=2; X=3; False
   - **False**

Zapytania atomic

1. Które z poniższych zapytań języka Prolog zakończą się spełnieniem celu:

   - ?- atomic("X").
   - **?- atomic('X').**
   - ?- atomic(X).
   - ?- atomic(_x_).
   - ?- atomic(_ma).
   - **?- atomic('ROK').**
   - ?- atomic("kok").
   - **?- atomic('_').**
   - ?- atomic(_).
   - ?- atomic("_").

2. Które z poniższych obiektów języka Prolog są stałymi:

   - "kok"
   - **'ROK' (prawda)**
   - _ma
   - X
   - **'x' (prawda)**

- "x"
- **x (prawda)**
- **'_' (prawda)**
- **'X' (prawda)**
- _
- "_"
- _x_

Typy danych

1. Stałe Symboliczne w języku Prolog
   - **mają charakter globalnych obiektów języka**
   - mają zasięg lokalny ograniczony wyłącznie do jednej reguły
   - mogą występować wyłącznie w zapytaniach albo w klauzulach, będących faktami
   - mają zasięg ograniczony wyłącznie do zbioru klauzul o tym samym nagłówku
2. Łańcuchy znakowe *(ang. string)* w języku Prolog:
   - mogą występować wyłącznie w zapytaniach albo klauzulach, będących faktami
   - **wymagają użycia ograniczników w postaci znaków apostrofu**
   - wymagają użycia ograniczników w postaci znaków cudzysłowu
   - mają zasięg lokalny ograniczony wyłącznie do jednej reguły
   - **zaliczane są do atomów (inaczej: stałych atomowych)**
   - **wymagają wcześniejszej deklaracji i określenia zasięgu wartości**
3. Identyfikator zmiennej w języku Prolog:
   - ma zasięg globalny
   - **jest lokalny względem pojedynczej klauzuli**
   - jest definiowany w momencie deklarowania dziedziny wartości zmiennej
   - jest lokalny względem zbioru klauzul o tym samym nagłówku
4. Wartości zmiennej w języku Prolog:
   - **są lokalne względem jednej instancji klauzuli**
   - **mają charakter globalny**
   - muszą być zgodne z zadeklarowanym wcześniej typem wartości
   - **są ustalane w procesie unifikacji (uzgadniania)**
5. Atomy w języku Prolog:
   - **obejmują stałe symboliczne oraz łańcuchy znakowe**

- mogą występować wyłącznie w zapytaniach albo w klauzulach będących faktami
- **mają charakter globalnych obiektów języka**
- mają zasięg lokalny ograniczony wyłącznie do jednej reguły
- wymagają wcześniejszej deklaracji i określenia zasięgu wartości

Agregujące

1. Jaki będzie wynik wykonania poniższego zapytania: ?- [X|Y]=[d,d,c,b,b,a], bagof(Z^X, member(Z,Y), W).:
   - W=[d^d,c^d,b^d,a^d]; False
   - W=[d^d]; W=[d^d]; W=[c^d]; W=[b^d]; W=[b^d]; W=[a^d]; False
   - **W=[d^d,c^d,b^d,b^d,a^d]; False**
   - W=[d^d]; W=[c^d]; W=[b^d]; W=[b^d]; W=[a^d]; False
2. Jaki będzie wynik wykonania poniższego zapytania:

?- [X|Y]=[d,d,c,b,b,a], setof(Z^X, member(Z,Y), W).:

   - W=[a^d]; W=[b^d]; W=[c^d]; W=[d^d]; False
   - **W=[a^d, b^d, c^d, d^d]; False**
   - W=[a^d]; W=[b^d]; W=[b^d]; W=[c^d]; W=[d^d]; W=[d^d]; False
   - W=[a^d, b^d, b^d, c^d, d^d, d^d]; False
3. Jaki będzie wynik wykonania poniższego zapytania: ?- [X|Y]=[a,b,c,d], bagof(X#Z, append(_, [Z|_], Y), W).:
   - W=[a#b, b#c, c#d]; False
   - W=[a#b]; W=[b#c]; W=[c#d]; False
   - **W=[a#b]; W=[a#c]; W=[a#d]; False**
   - W=[a#b]; False
4. Jaki będzie wynik wykonania poniższego zapytania: ?- [X|Y]=[a,b,c,d], bagof(X^Z, append(_, [Z|_], [X|Y]), W).
   - W=[a^b, a^c, a^d]; False
   - W=[a^b]; W=[a^c]; W=[a^d]; False
   - W=[a^a, a^b, a^c, a^d]; False
   - **W=[a^a]; W=[a^b]; W=[a^c]; W=[a^d]; False**
5. Jaki będzie wynik wykonania poniższego zapytania: ?- [X|Y]=[a,b,c,d], findall(X^Z, append(_, [Z,_|_], [X|Y]), W).
   - W=[a^b, c^d]
   - W=[a^a, a^b, a^c, a^d]
   - W=[a^b, a^c, a^d]
   - **W=[a^a, a^b, a^c]**

Nawroty z odcięciami

1. Dla następującego programu prologowego: p(N):-N=1; N=2,!; N=3,!. , które z poniższych odpowiedzi są wszystkimi rozwiązaniami dla zapytania ?- p(X),!,p(Y). :

   – X=1 Y=1

   – X=1 Y=1; X=2 Y=2

   – **X=1 Y=1; X=1 Y=2**

   – X=1 Y=1; X=1 Y=2; X=1 Y=3

2. Dla następującego programu prologowego: p(1). p(2). p(3):-!., które z poniższych odpowiedzi są wszystkimi rozwiązaniami dla zapytania ?- p(X),!,p(Y). :

   – X=1 Y=1

   – **X=1 Y=1; X=1 Y=2; X=1 Y=3**

   – X=1 Y=1; X=1 Y=2

   – X=1 Y=1; X=1 Y=2; X=1 Y=3;
     X=2 Y=1; X=2 Y=2; X=2 Y=3;
     X=3 Y=1; X=3 Y=2; X=3 Y=3;

3. Dla następującego programu prologowego: p(1). p(2):-!. p(3):-!., które z poniższych odpowiedzi są wszystkimi rozwiązaniami dla zapytania ?- p(X),p(Y),!. :

   – X=1 Y=1; X=1 Y=2; X=2 Y=1; X=2 Y=2

   – X=1 Y=1; X=2 Y=2

   – **X=1 Y=1**

   – X=1 Y=1; X=1 Y=2; X=1 Y=3; X=2 Y=1; X=2 Y=2; X=2 Y=3; X=3 Y=1; X=3 Y=2; X=3 Y=3;

   – X=1 Y=1; X=1 Y=2; X=2 Y=1; X=2 Y=2

4. Dla następującego programu prologowego: p(N):-N=1; N=2; N=3, !. , które z poniższych odpowiedzi są wszystkimi rozwiązaniami dla zapytania ?- !, p(X), p(Y). :

   – X=1 Y=1

   – X=1 Y=1; X=1 Y=2; X=1 Y=3

   – **X=1 Y=1; X=1 Y=2; X=1 Y=3; X=2 Y=1; X=2 Y=2; X=2 Y=3; X=3 Y=1; X=3 Y=2; X=3 Y=3;**

   – X=1 Y=1; X=2 Y=2; X=3 Y=3

5. Dla następującego programu prologowego: p(N):-N=1; N=2, !; N=3, !. , które z poniższych odpowiedzi są wszystkimi rozwiązaniami dla zapytania ?- !, p(X), p(Y). :

   – X=1 Y=1

   – X=1 Y=1; X=2 Y=2

   – **X=1 Y=1; X=1 Y=2; X=2 Y=1; X=2 Y=2**

   – X=1 Y=1; X=1 Y=2; X=1 Y=3; X=2 Y=1; X=2 Y=2; X=2 Y=3; X=3 Y=1; X=3 Y=2; X=3 Y=3;

6. Dla następującego programu prologowego: p(1):-!. p(2):-!. p(3). ,które z poniższych odpowiedzi są wszystkimi rozwiązaniami zapytania ?- p(X),!,p(Y).:

- X=1 Y=1; X=2 Y=2
- **X=1 Y=1**
- X=1 Y=1; X=1 Y=2
- X=1 Y=1; X=1 Y=2; X=1 Y=3

7. Biorąc pod uwagę następujące definicje: smutny(X) :- not(szczęśliwy(X)). szczęśliwy(X) :- piękny(X), bogaty(X). bogaty(król). bogaty(książe). piękny(książe). piękny(śnieżka).
   - ?- smutny(król). **true**/false
   - ?- smutny(śnieżka). **true**/false
   - ?- smutny(książe). true/**false**
   - ?- smutny(królowa). **true**/false
   - ?- smutny(**K**toś). true/**false**
   - ?- smutny(**k**toś). **true**/false (mała litera)