§3 Nielogiczne elementy Prologu

"elementy Prologu, który zniszcza resolucję"

3.1 Arytmetyka

```
length([],0).
length([_|L],N):- length(L,M), N = M+1.
```

```
?- length([1,2,3],N).
   N = ((0+1)+1)+1        unifikacja!
```

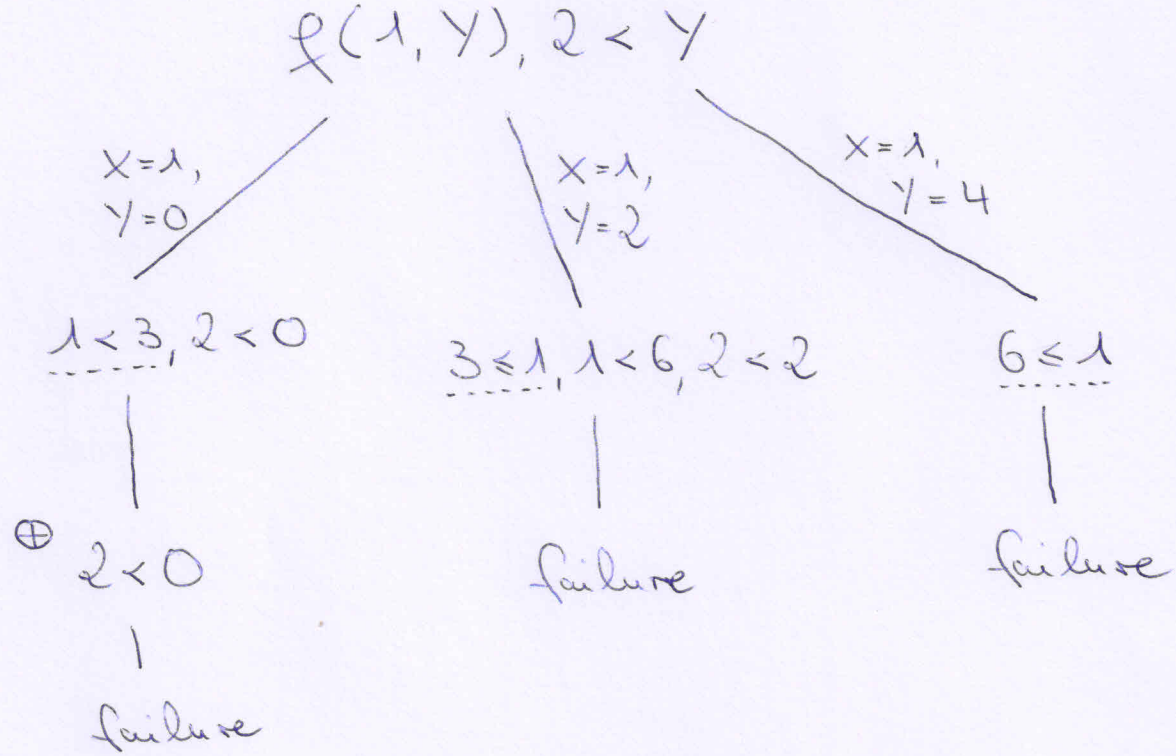
```
?- X = Y+1, 2 = 1+2.
   X = Y+1, 2 = 1+2.
?- 2 = 2, X = Z+Y.
   X = 2+Y.
```

is wymusza obliczenie,
ale zmienne na prawej stronie już muszą mieć wartości:

```
? 2 is 1+2.
  2 = 3
? 4 is 2+1.
  false
?- 2 is X+1.   ↯
```

3.2 kontrolowanie Backtrackingu

```
f(X,0) :- X < 3.
f(X,2) :- 3 ≤ X, X < 6.
f(X,4) :- 6 ≤ X.
```



uwaga: $1 < 3$ spełnione

$\rightsquigarrow$ $3 \leq 1$ i $6 \leq 1$ nie spełnione

czyli już w momencie ⊕ wiadomo, że albo $2 < 0$ prawdą albo ostateczna odpowiedź będzie false.

↳ cut

```
f(X,0) :- X < 3, !.
f(X,2) :- 3 ≤ X, X < 6, !.
f(X,4) :- 6 ≤ X.
```

Po cut'u Prolog ignoruje pozostałych reguł.

↳ optimalizacja:

```
f̄(X,0) :- X < 3, !.
f̄(X,2) :- X < 6, !.
f̄(X,4).
```

uwaga: ewtl. zmieni się semantyka programu

```
?- f(1,Y).
   Y = 0
```

$\bar{f}$ bez cut'u:

```
?- f̄(1,Y).
   Y = 0;
   Y = 2;
   Y = 4
```

"red cut" i "green cut"

! eliminuje pewne rezolucje SLD z drzewa odpowiedzi.

Def. Cut

(i) ! zawsze jest spełniony.

(ii) Jeżeli reguła $H :- B_1, \ldots B_m, !, B_{m+1}, \ldots B_n$. zostaje zastosowana przez jakiś cel G (mówimy, że G, to "parent goal"), to po dojściu do ! już nie wolno zmienić wartości zmiennych, czyli substytucję σ, która unifikowała G i H oraz rozwiązała $B_1, \ldots B_m$.

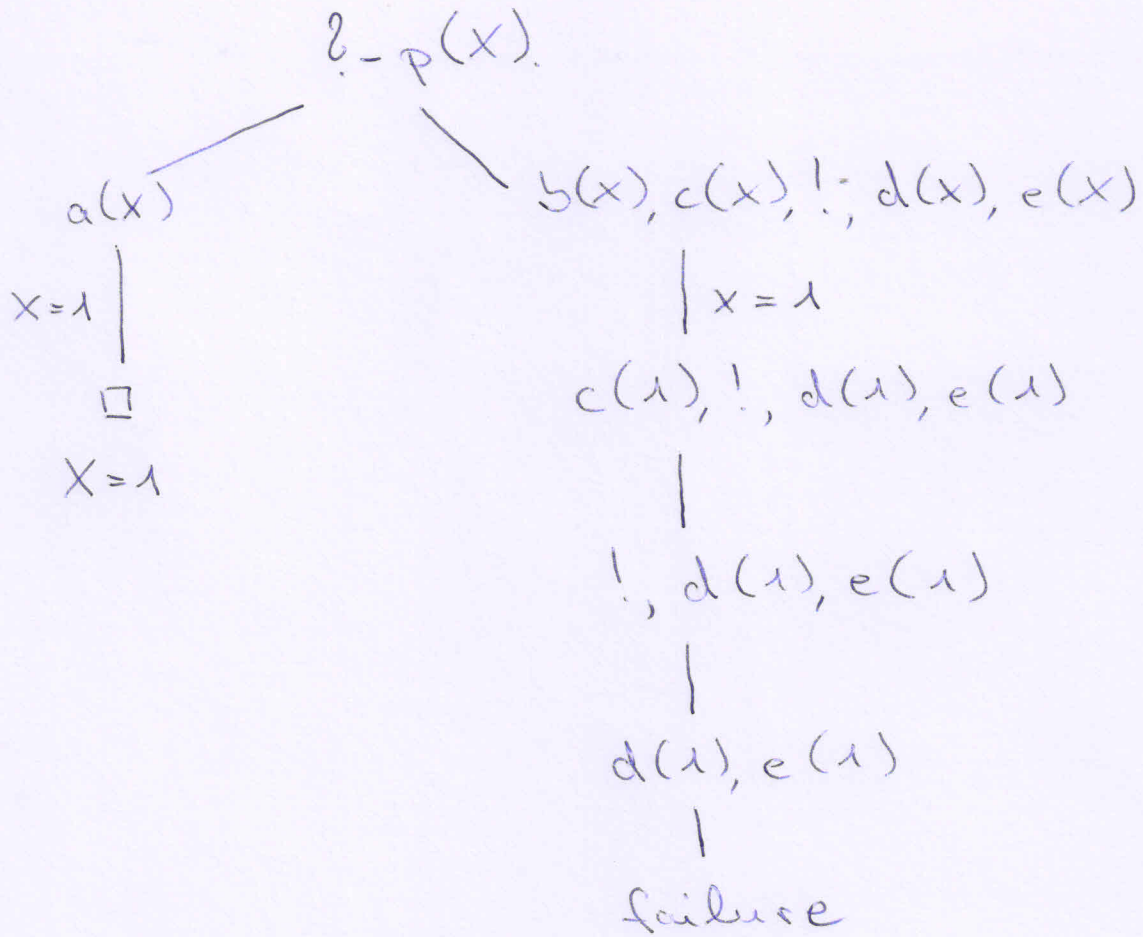
(iii) Więc też nie wolno używać dalsze klauzuli, czyli jedyne rozwiązanie dla G, to $\sigma(B_{m+1}), \ldots \sigma(B_n)$.
Uwaga: "Nowe" wartości w $B_{m+1}, \ldots B_n$ wolno zmienić.

Przykład:

```
p(X) :- a(X).
p(X) :- b(X), c(X), !, d(X), e(X).
p(X) :- f(X).
a(1). b(1). b(2). c(1). c(2). d(2). e(2). f(3).
```

bez !:

```
?- p(X).
X = 1;
X = 2;
X = 3;
false
```

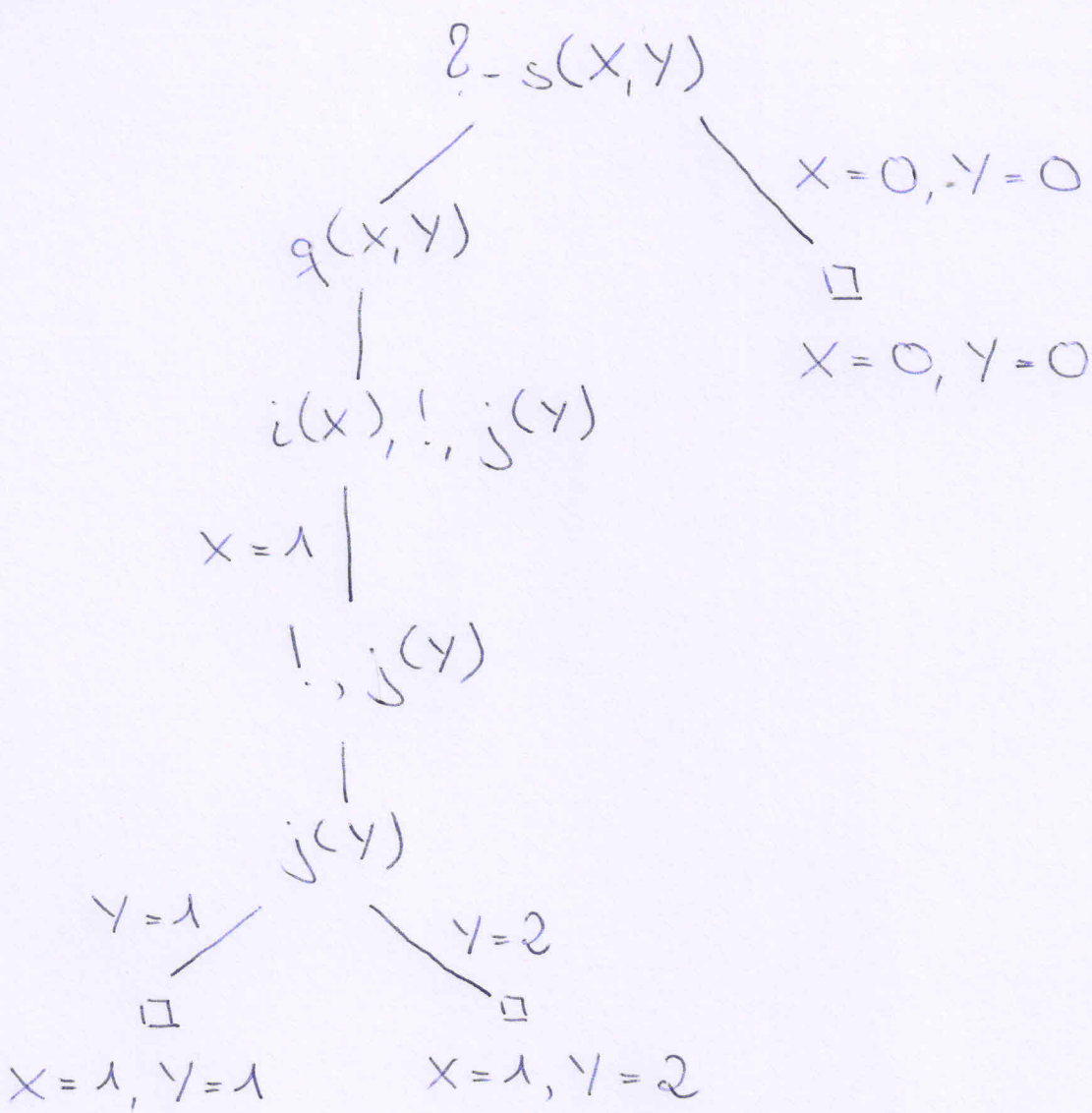


- Przykład:

```
s(X,Y) :- q(X,Y).
s(0,0).
q(X,Y) :- i(X), !, j(Y).
i(1). j(1). i(2). j(2).
```

bez !: ?- s(X,Y).

X = 1, Y = 1;
X = 1, Y = 2;
X = 2, Y = 1;
X = 2, Y = 2;
X = 0, Y = 0;
false



Przykład:

(i)
```
max(X,Y,X) :- X >= Y.
max(X,Y,X) :- X < Y.
```
↳
```
max(X,Y,X) :- X >= Y, !.
max(X,Y,Y).
```

(ii) Dodawanie elementu X do listy L, tylko dla $x \notin L$:

```
dod(X,L,L) :- member(X,L).
dod(X,L,[X|L]).
```

```
?- dod(b,[a,b,c],L).
   L = [a,b,c];
   L = [b,a,b,c]
```

Problem: Nie ma negacji na prawych stronach reguł.

↳
```
add(X,L,L) :- member(X,L),!.
add(X,L,[X|L]).
```

3.3 Negation as failure

"Mary likes all animals, but snakes."

```
likes(mary,X) :- snake(X),!,fail.
likes(mary,X) :- animal(X).
```

```
not(P) :- P,!,fail.
not(P).
```

```
likes(mary,X) :- animal(X), \+ snake(X).
```

Uwaga:
Prolog nie pokazuje $\neg P$,
tzn. \+ P nie jest negacją w sensie logicznym.

\+ P jest spełnione, jeżeli nie uda się pokazywać P (czyli jeżeli "nie ma wystarczającej informacji" dla P).

↳ Closed World Assumption

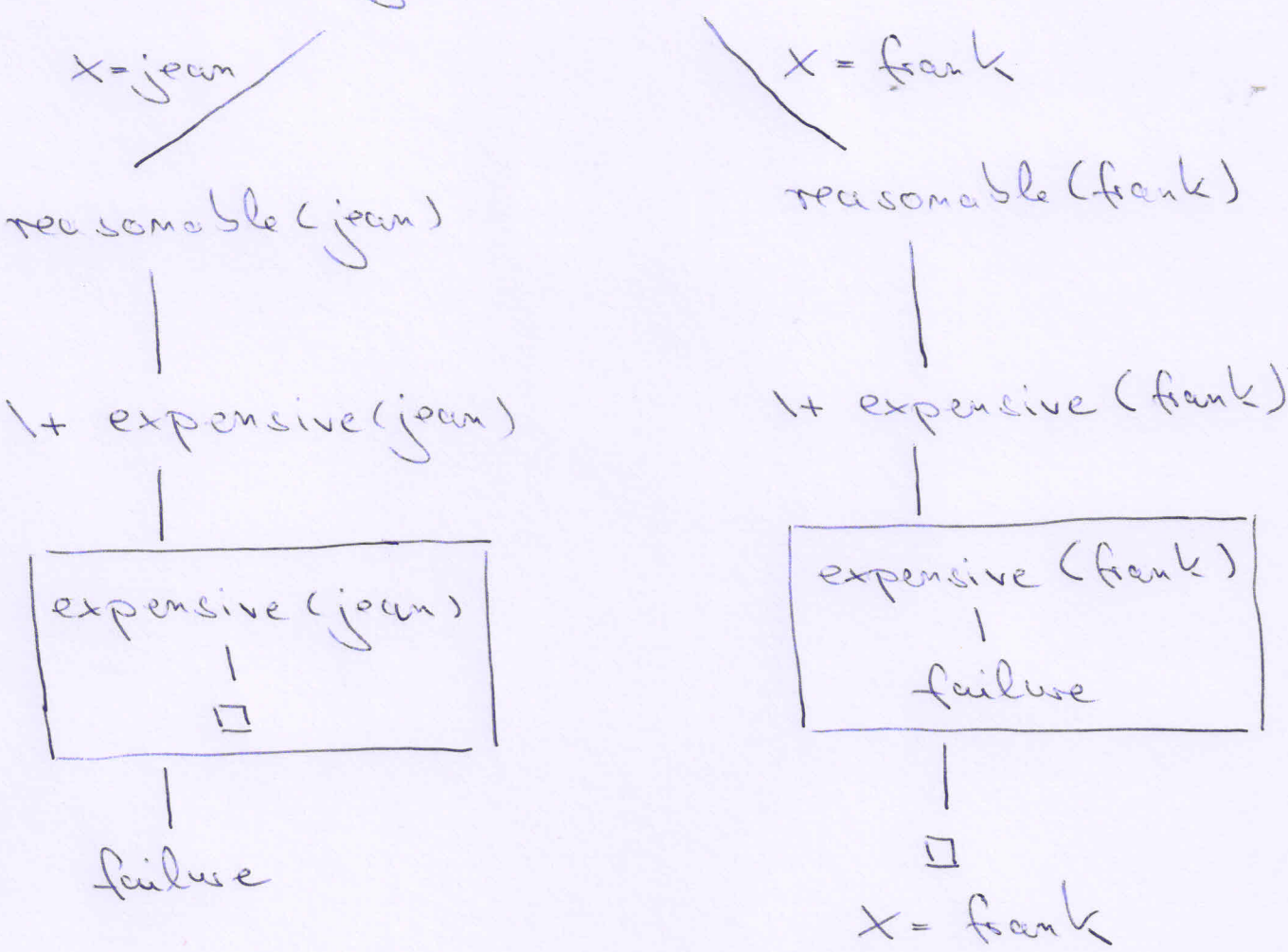
Konsekwencja: Nie ma monotoniczności

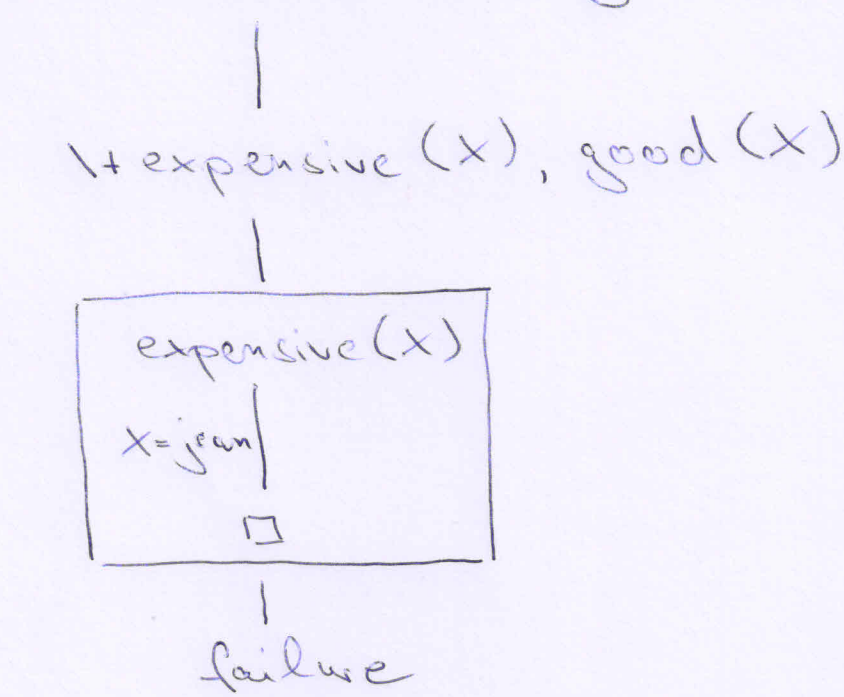
```
p(X) :- \+ r(X).
r(a).

?- p(b).
true
```

```
p(X) :- \+ r(X).
r(a).
r(b).

?- r(b).
false
```

Przykład:

```
good(jean).
expensive(jean).
good(frank).
reasonable(Bar) :- \+ expensive(Bar).
```





$\forall x : \neg \text{expensive}(x) \models \neg \exists x : \text{expensive}(x)$