Um fragmento da teoria das palavras:

1. $\forall x\, substr(x, x)$
2. $\forall x \forall y \forall z\, [(substr(x, y) \wedge suffix(y, z)) \rightarrow substr(x, z)]$
3. $\forall x \forall y\, suffix(x, y.x)$
4. $\forall x \forall y \forall z\, [(substr(x, y) \wedge prefix(y, z)) \rightarrow substr(x, z)]$
5. $\forall x \forall y\, prefix(x, x.y)$

Na forma clausal:

1. $substr(x, x)$
2. $\neg substr(x, y) \vee \neg suffix(y, z) \vee substr(x, z)$
3. $suffix(x, y.x)$
4. $\neg substr(x, y) \vee \neg prefix(y, z) \vee substr(x, z)$
5. $prefix(x, x.y)$

Uma refutação de $\neg substr(a.b.c, a.a.b.c.c)$:

6.	$\neg substr(a.b.c, a.a.b.c.c)$	neg. da consulta
7.	$\neg substr(a.b.c, y_1) \vee \neg suffix(y_1, a.a.b.c.c)$	6,2
8.	$\neg substr(a.b.c, a.b.c.c)$	7,3
9.	$\neg substr(a.b.c, y_2) \vee \neg prefix(y_2, a.b.c.c)$	8,4
10.	$\neg substr(a.b.c, a.b.c)$	9,5
11.	$\square$	10,1

SLD-resolution:

Sejam:

P: conjunto de cláusulas de programa;

R: regra de computação;

G: cláusula meta.

Uma *derivação por SLD-resolution* é uma sequência de passos de resolução entre cláusulas meta e cláusulas de programa, levando a uma sequência de cláusulas meta $G_0, G_1, \ldots$, onde:

G_0 é G; e

G_{i+1} é obtido de G_i:

- selecionando-se um literal A_i de G_i de acordo com R;
- escolhendo-se uma cláusula C_i de P cuja cabeça unifica com A_i com umg θ_i;
- e resolvendo:
  $G_i = \leftarrow A_1, \ldots, A_{i-1}, A_i, A_{i+1}, \ldots, A_n$
  $C_i = A \leftarrow B_1, \ldots, B_k$
  $A_i\theta_i = A\theta_i$
  $G_{i+1} = \leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_k, A_{i+1}, \ldots, A_n)\theta_i$

Exemplo:

Dois procedimentos e um banco de dados:

1. $q(x, y) \leftarrow p(x, y)$
2. $q(x, y) \leftarrow p(x, z), q(z, y)$

3. $p(b, a)$	7. $p(f, b)$
4. $p(c, a)$	8. $p(h, g)$
5. $p(d, b)$	9. $p(i, h)$
6. $p(e, b)$	10. $p(h, j)$

O mesmo programa em Prolog:

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).

parent(bob,allen).
parent(catherine,allen).
parent(dave,bob).
parent(ellen,bob).
parent(fred,bob).
parent(harry,george).
parent(ida,harry).
parent(harry,joe).
```

Uma consulta:

```
:- ancestor(Y,bob), ancestor(bob,Z).
```

Resposta: `Y = dave, Z = allen.`