

# Chapter 6

## Negation: Procedural Interpretation

# Outline

- Motivate negation with two examples
- Extended programs and queries
- The computation mechanism: SLDNF-derivations
- Allowed programs and queries
- Negation in Prolog

[Apt and Bol, 1994]

Krzysztof Apt and Roland Bol. Logic Programming and Negation: A Survey.  
*Journal of Logic Programming*, 19/20:9-71, 1994.

# Why Negation? Example (I)

```
attend(flp, andreas) ←  
attend(flp, maja) ←  
attend(flp, dirk) ←  
attend(flp, natalia) ←  
attend(fcp, andreas) ←  
attend(fcp, maja) ←  
attend(fcp, stefan) ←  
attend(fcp, arturo) ←
```

Who attends FCP but not FLP?

*attend(fcp, x), ¬attend(flp, x)*

## Why Negation? Example (II)

sets (lists)  $A = [a_1, \dots, a_m]$  and  $B = [b_1, \dots, b_n]$  disjoint

$:\Leftrightarrow$

- $m = 0$ , or
- $m > 0$ ,  $a_1 \notin B$ , and  $[a_2, \dots, a_m]$  and  $B$  are disjoint

*disjoint*([ ],  $x$ )  $\leftarrow$

*disjoint*([ $x|y$ ],  $z$ )  $\leftarrow \neg member(x, z), disjoint(y, z)$

# Extended Logic Programs and Queries

- “ $\neg$ ” negation sign
- $A, \neg A$  **literals**  $:\Leftrightarrow A$  atom
- $A, \neg A$  ground literals  $:\Leftrightarrow A$  ground atom
- **(extended) query**  $:\Leftrightarrow$  finite sequence of literals
- $H \leftarrow \underline{B}$  **(extended) clause**  
 $:\Leftrightarrow H$  atom,  $\underline{B}$  extended query
- **(extended) program**  
 $:\Leftrightarrow$  finite set of extended clauses

# How do we Compute?

Negation as Failure (NF) : $\Leftrightarrow$

1. Suppose  $\neg A$  is selected in the query  $Q = \underline{L}, \neg A, \underline{N}$ .
2. If  $P \cup \{A\}$  succeeds, then the derivation of  $P \cup \{Q\}$  fails at this point.
3. If all derivations of  $P \cup \{A\}$  fail, then  $Q$  resolves to  $Q' = \underline{L}, \underline{N}$ .

$\neg A$  succeeds iff  $A$  finitely fails.

$\neg A$  finitely fails iff  $A$  succeeds.

**SLDNF** = Selection rule driven Linear resolution for Definite clauses augmented by Negation as Failure rule

# SLDNF-Resolvents

1.  $Q = \underline{L}, A, \underline{N}$  query;  $A$  selected, positive literal
  - $H \leftarrow \underline{M}$  variant of a clause  $c$  which is variable-disjoint with  $Q$ ,  $\theta$  MGU of  $A$  and  $H$
  - $Q' = (\underline{L}, \underline{M}, \underline{N})\theta$  **SLDNF-resolvent** of  $Q$  (and  $c$  w.r.t.  $A$  with  $\theta$ )
  - We write this SLDNF-derivation step as  $Q \xRightarrow[c]{\theta} Q'$
2.  $Q = \underline{L}, \neg A, \underline{N}$  query;  $\neg A$  selected, negative **ground** literal
  - $Q' = \underline{L}, \underline{N}$  **SLDNF-resolvent** of  $Q$  (w.r.t.  $\neg A$  with  $\epsilon$ )
  - We write this SLDNF-derivation step as  $Q \xRightarrow[\epsilon]{\theta} Q'$

# Pseudo Derivations

A maximal sequence of SLDNF-derivation steps

$$Q_0 \xRightarrow[c_1]{\theta_1} Q_1 \dots Q_n \xRightarrow[c_{n+1}]{\theta_{n+1}} Q_{n+1} \dots$$

is a **pseudo derivation** of  $P \cup \{Q_0\} : \Leftrightarrow$

- $Q_0, \dots, Q_{n+1}, \dots$  are queries, each empty or with one literal selected in it;
- $\theta_1, \dots, \theta_{n+1}, \dots$  are substitutions;
- $c_1, \dots, c_{n+1}, \dots$  are clauses of program  $P$  (in case a positive literal is selected in the preceding query);
- for every SLDNF-derivation step with input clause “standardization apart” holds.



# Forests

$\mathcal{F} = (\mathcal{T}, T, subs)$  forest  $:\Leftrightarrow$

- $\mathcal{T}$  set of trees where
  - nodes are queries;
  - a literal is selected in each non-empty query;
  - leaves may be marked as “success”, “failure”, or “floundered”.
- $T \in \mathcal{T}$  main tree
- $subs$  assigns to some nodes of trees in  $\mathcal{T}$  with selected negative ground literal  $\neg A$  a subsidiary tree of  $\mathcal{T}$  with root  $A$ .

tree  $T \in \mathcal{T}$  successful  $:\Leftrightarrow$  it contains a leaf marked as “success”

tree  $T \in \mathcal{T}$  finitely failed  $:\Leftrightarrow$  it is finite and all leaves are marked as “failure”

# Pre-SLDNF-Trees

The class of **pre-SLDNF-trees** for a program  $P$  is the smallest class  $\mathcal{C}$  of forests such that

- for every query  $Q$ :  
the initial pre-SLDNF-tree  $(\{T_Q\}, T_Q, subs)$  is in  $\mathcal{C}$ , where  $T_Q$  contains the single node  $Q$  and  $subs(Q)$  is undefined
- for every  $\mathcal{F} \in \mathcal{C}$ :  
the extension of  $\mathcal{F}$  is in  $\mathcal{C}$

# Extension of Pre-SLDNF-Tree (I)

**extension** of  $\mathcal{F} = (\mathcal{T}, T, subs) : \Leftrightarrow$

1. Every occurrence of the empty query is marked as “**success**”.
2. For every non-empty query  $Q$ , which is an unmarked leaf in some tree in  $\mathcal{T}$ ,

perform the following action:

Let  $L$  be the selected literal of  $Q$ .

- $L$  positive.
  - $Q$  has no SLDNF-resolvents  
 $\Rightarrow Q$  is marked as “**failure**”
  - else  
 $\Rightarrow$  for every program clause  $c$  which is applicable to  $L$ , exactly one direct descendant of  $Q$  is added. This descendant is an SLDNF-resolvent of  $Q$  and  $c$  w.r.t.  $L$ .

## Extension of Pre-SLDNF-Tree (II)

- $L = \neg A$  negative.
  - $A$  non-ground  $\Rightarrow$   $Q$  is marked as “floundered”
  - $A$  ground
    - \*  $subs(Q)$  undefined
      - $\Rightarrow$  new tree  $T'$  with single node  $A$  is added to  $\mathcal{T}$  and  $subs(Q)$  is set to  $T'$
    - \*  $subs(Q)$  defined and successful
      - $\Rightarrow$   $Q$  is marked as “failure”
    - \*  $subs(Q)$  defined and finitely failed
      - $\Rightarrow$  SLDNF-resolvent of  $Q$  is added as the only direct descendant of  $Q$
    - \*  $subs(Q)$  defined and neither successful nor finitely failed
      - $\Rightarrow$  no action

# SLDNF-Trees

## SLDNF-tree

$:\Leftrightarrow$  limit of a sequence  $\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2, \dots$ , where

- $\mathcal{F}_0$  initial pre-SLDNF-tree
- $\mathcal{F}_{i+1}$  extension of  $\mathcal{F}_i$ , for every  $i \in \mathbb{N}$

SLDNF-tree for  $P \cup \{Q\}$

$:\Leftrightarrow$

SLDNF-tree in which  $Q$  is the root of the main tree

# Successful, Failed, and Finite SLDNF-Trees

(pre-)SLDNF-tree **successful**

: $\Leftrightarrow$  its main tree is successful

(pre-)SLDNF-tree **finitely failed**

: $\Leftrightarrow$  its main tree is finitely failed

SLDNF-tree **finite**

: $\Leftrightarrow$  no infinite paths exist in it,

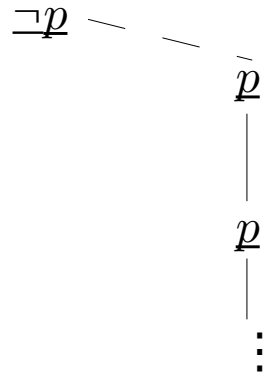
where a path is a sequence of nodes  $N_0, N_1, N_2, \dots$  such that for every  $i = 0, 1, 2, \dots$ :

- either  $N_{i+1}$  is a direct descendant of  $N_i$
- or  $N_{i+1}$  is the root of  $subs(N_i)$ .

# Example (I)

$$p \leftarrow p$$

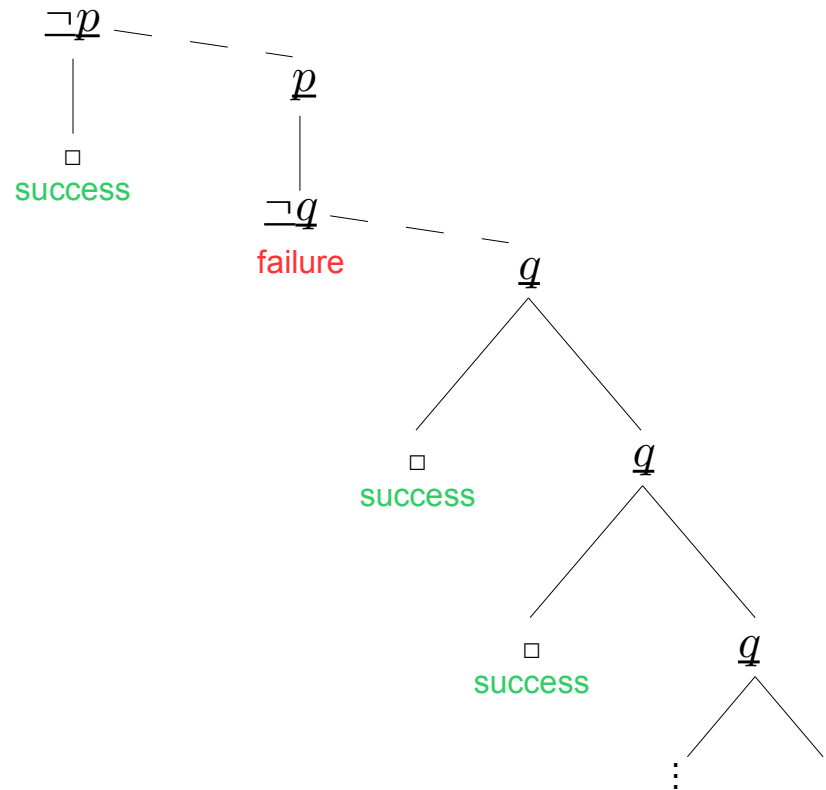
SLDNF-tree for  $P \cup \{\neg p\}$  is infinite:



# Example (II)

$p \leftarrow \neg q$   
 $q \leftarrow$   
 $q \leftarrow q$

SLDNF-tree for  $P \cup \{\neg p\}$  is successful:





# SLDNF-Derivation

SLDNF-derivation of  $P \cup \{Q\} : \Leftrightarrow$

branch in the main tree of an SLDNF-tree  $\mathcal{F}$  for  $P \cup \{Q\}$  together with the set of all trees in  $\mathcal{F}$  whose roots can be reached from the nodes in this branch

SLDNF-derivation **successful** :  $\Leftrightarrow$

it ends with  $\square$

Let the main tree of an SLDNF-tree for  $P \cup \{Q_0\}$  contain a branch

$$\xi = Q_0 \xRightarrow{\theta_1} Q_1 \dots Q_{n-1} \xRightarrow{\theta_n} Q_n = \square :$$

computed answer substitution (CAS) of  $Q_0$  (w.r.t.  $\xi$ ) :  $\Leftrightarrow$

$$(\theta_1 \cdots \theta_n) \upharpoonright_{\text{Var}(Q_0)}$$

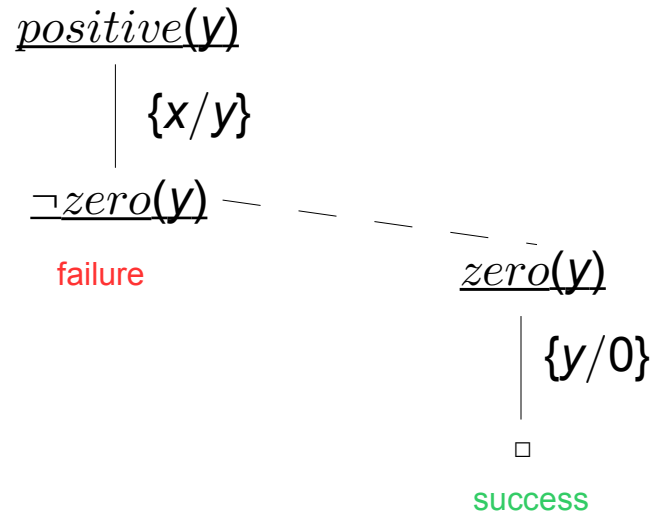
# A Theorem on Limits

Theorem 3.10 ([Apt and Bol, 1994])

- (i) Every SLDNF-tree is the limit of a unique sequence of pre-SLDNF-trees.
- (ii) If the SLDNF-tree  $\mathcal{F}$  is the limit of the sequence  $\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2, \dots$ , then:
  - a)  $\mathcal{F}$  is successful and yields CAS  $\theta$   
iff some  $\mathcal{F}_i$  is successful and yields CAS  $\theta$ ,
  - b)  $\mathcal{F}$  finitely failed  
iff some  $\mathcal{F}_i$  is finitely failed.

# Why Only Select Negative Literals if they are Ground? (I)

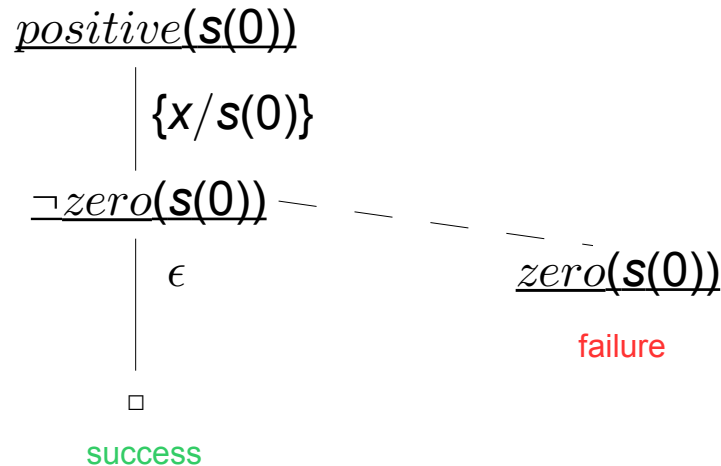
$c_1:$      $zero(0) \leftarrow$   
 $c_2:$      $positive(x) \leftarrow \neg zero(x)$



Hence,  $\neg \exists y \text{ positive}(y)?$ , i.e.  $\forall y \neg \text{positive}(y)?$

# Why Only Select Negative Literals if they are Ground? (II)

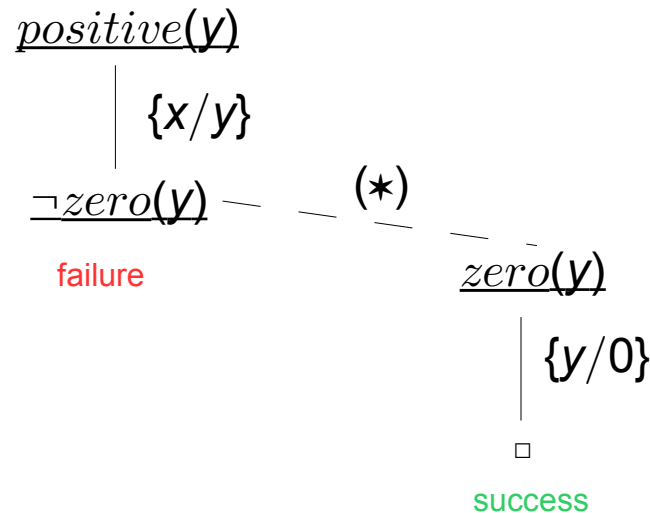
$c_1:$	$zero(0) \leftarrow$
$c_2:$	$positive(x) \leftarrow \neg zero(x)$



Hence,  $positive(s(0))!$ , i.e.  $\exists y positive(y)!$

# Why Only Select Negative Literals if they are Ground? (III)

$C_1:$      $zero(0) \leftarrow$   
 $C_2:$      $positive(x) \leftarrow \neg zero(x)$



Fundamental mistake in (\*):  $\exists y \ zero(y)$  is not the opposite of  $\exists y \ \neg zero(y)$

# Selection of Non-Ground Negative Literals in Prolog

```
zero(0).  
positive(X) :- \+ zero(X).
```

```
| ?- positive(0).  
no
```

```
| ?- positive(s(0)).  
yes
```

```
| ?- positive(Y).  
no
```

# Extended Selection Rules

(extended) selection rule  $:\Leftrightarrow$

function which, given a pre-SLDNF-tree  $\mathcal{F} = (\mathcal{T}, T, subs)$ , selects a literal in every non-empty unmarked leaf in every tree in  $\mathcal{T}$ .

SLDNF-tree  $\mathcal{F}$  is according to selection rule  $\mathcal{R} :\Leftrightarrow$

$\mathcal{F}$  is the limit of a sequence of pre-SLDNF-trees in which literals are selected according to  $\mathcal{R}$ .

selection rule  $\mathcal{R}$  is safe  $:\Leftrightarrow$

$\mathcal{R}$  never selects a non-ground negative literal

# Blocked Queries

query  $Q$  **blocked**

$:\Leftrightarrow$

$Q$  non-empty and contains exclusively non-ground negative literals

$P \cup \{Q\}$  **flounders**

$:\Leftrightarrow$

some SLDNF-tree for  $P \cup \{Q\}$  contains a blocked node



# Allowed Programs and Queries

query  $Q$  **allowed**

$:\Leftrightarrow$

every  $x \in \text{Var}(Q)$  occurs in a positive literal of  $Q$

clause  $H \leftarrow \underline{B}$  **allowed**  $:\Leftrightarrow \neg H, \underline{B}$  allowed

(thus: unit clause  $H \leftarrow$  allowed  $:\Leftrightarrow H$  ground atom)

program  $P$  **allowed**  $:\Leftrightarrow$  all its clauses are allowed

# Allowed Programs and Queries do not Flounder

Theorem 3.13 ([Apt and Bol, 1994])

Suppose that  $P$  and  $Q$  are allowed. Then,

- (i)  $P \cup \{Q\}$  does not flounder;
- (ii) if  $\theta$  is a CAS of  $Q$ , then  $Q\theta$  is ground.

# An Example

```
zero(0) ←  
positive(x) ← ¬zero(x)
```

This program is not allowed.

```
zero(0) ←  
positive(x) ← num(x), ¬zero(x)  
num(0) ←  
num(s(x)) ← num(x)
```

This program is allowed.

# Specifics of PROLOG

- Leftmost selection rule  
LDNF-resolution, LDNF-resolvent, LDNF-tree, ...
- **Non-ground negative literals are selected!**
- A program is a sequence of clauses
- Unification without occur check
- Depth-first search, backtracking

# Extended Prolog Trees

Let  $P$  extended program and  $Q_0$  extended query.

**Extended Prolog** Tree for  $P \cup \{Q_0\}$  is forest of finitely branching, ordering trees of queries, possibly marked with “success” or “failure”, produced as follows:

- Start with forest  $(\{T_{Q_0}\}, T_{Q_0}, \text{subs})$ , where  $T_{Q_0}$  contains the single node  $Q_0$  and  $\text{subs}(Q_0)$  is undefined
- Repeatedly apply to current forest  $\mathcal{F} = (\mathcal{T}, T, \text{subs})$  and leftmost unmarked leaf  $Q$  in  $T_1$ , where  $T_1 \in \mathcal{T}$  is leftmost, bottommost (=most nested subsidiary) tree with an unmarked leaf, the operation *expand*( $\mathcal{F}, Q$ )

# Operation Expand (I)

operation *expand*( $\mathcal{F}$ ,  $Q$ ) is defined by:

- if  $Q = \square$ , then
  1. mark  $Q$  with “success”
  2. if  $T_1 \neq T$ , then remove from  $T_1$  all edges to the right of the branch that ends with  $Q$
- if  $Q$  has no LDNF-resolvents, then mark  $Q$  with “failure”
- else let  $L$  be the leftmost literal in  $Q$ :

## Operation Expand (II)

- $L$  is positive:  
add for each clause that is applicable to  $L$  an LDNF-resovent as descendant of  $Q$   
(such that the order of the clauses is respected)
- $L = \neg A$  is negative (not necessarily ground):
  - \* if  $subs(Q)$  is undefined,  
then add a new tree  $T' = A$  and set  $subs(Q)$  to  $T'$
  - \* if  $subs(Q)$  is defined and successful,  
then mark  $Q$  with “failure”
  - \* if  $subs(Q)$  is defined and finitely failed,  
then add in  $T_1$  the LDNF-resolvent of  $Q$  as the only descendant of  $Q$

# Operation Expand (III)

operation *expand*( $\mathcal{F}$ ,  $Q$ ) is defined by:

- if  $Q = \square$ , then
  1. mark  $Q$  with “success”
  2. if  $T_1 \neq T$ , then remove from  $T_1$  all edges to the right of the branch that ends with  $Q$
- if  $Q$  has no LDNF-resolvents, then mark  $Q$  with “failure”
- else let  $L$  be the leftmost literal in  $Q$ :
  - $L$  is positive:

add for each clause that is applicable to  $L$  an LDNF-resolvent as descendant of  $Q$   
(such that the order of the clauses is respected)
  - $L = \neg A$  is negative (not necessarily ground):
    - \* if  $subs(Q)$  is undefined, then add a new tree  $T' = A$  and set  $subs(Q)$  to  $T'$
    - \* if  $subs(Q)$  is defined and successful, then mark  $Q$  with “failure”
    - \* if  $subs(Q)$  is defined and finitely failed,

then add in  $T_1$  the LDNF-resolvent of  $Q$  as the only descendant of  $Q$



# Floundering is Ignored (I)

```
even(0).  
even(X) :- \+ odd(X).  
odd(s(X)) :- even(X).
```

```
| ?- even(X).
```

```
X = 0 ;
```

```
no
```

```
| ?- even(s(s(0))).
```

```
yes
```

## Floundering is Ignored (II)

```
num(0).  
num(s(X)) :- num(X).  
even(X) :- num(X), \+ odd(X).  
odd(s(X)) :- even(X).
```

```
| ?- even(X).
```

```
X = 0 ;
```

```
X = s(s(0)) ;
```

```
X = s(s(s(s(0)))) ;
```

```
⋮
```

# Objectives

- Motivate negation with two examples
- Extended programs and queries
- The computation mechanism: SLDNF-derivations
- Allowed programs and queries
- Negation in Prolog