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)

$attend(flp, andreas) \leftarrow$
$attend(flp, maja) \leftarrow$
$attend(flp, dirk) \leftarrow$
$attend(flp, natalia) \leftarrow$
$attend(fcp, andreas) \leftarrow$
$attend(fcp, maja) \leftarrow$
$attend(fcp, stefan) \leftarrow$
$attend(fcp, arturo) \leftarrow$

Who attends FCP but not FLP?

$$attend(fcp, x), \neg attend(flp, x)$$

Why Negation? Example (II)

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

$:\Leftrightarrow$

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

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

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

Extended Logic Programs and Queries

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

How do we Compute?

Negation as Failure (NF) :⇔

1. Suppose $\neg A$ is selected in the query $Q = \underline{\underline{L}}, \neg A, \underline{\underline{N}}$.
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 $Q' = \underline{\underline{L}}, \underline{\underline{N}}$.

$\neg A$ 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. $Q = \underline{\underline{L}}, A, \underline{\underline{N}}$ query; A selected, positive literal

- $H \leftarrow \underline{\underline{M}}$ variant of a clause c which is variable-disjoint with Q, θ MGU of A and H
- $Q' = (\underline{\underline{L}}, \underline{\underline{M}}, \underline{\underline{N}})\theta$ SLDNF-resolvent of Q (and c w.r.t. A with θ)
- We write this SLDNF-derivation step as $Q \overset{\theta}{\underset{c}{\Longrightarrow}} Q'$

2. $Q = \underline{\underline{L}}, \neg A, \underline{\underline{N}}$ query; $\neg A$ selected, negative ground literal

- $Q' = \underline{\underline{L}}, \underline{\underline{N}}$ SLDNF-resolvent of Q (w.r.t. $\neg A$ with ϵ)
- We write this SLDNF-derivation step as $Q \overset{\theta}{\underset{\epsilon}{\Longrightarrow}} Q'$

Pseudo Derivations

A maximal sequence of SLDNF-derivation steps

$$Q_0 \overset{\theta_1}{\underset{c_1}{\Longrightarrow}} Q_1 \ldots Q_n \overset{\theta_{n+1}}{\underset{c_{n+1}}{\Longrightarrow}} Q_{n+1} \ldots$$

- is a pseudo derivation of $P \cup \{Q_0\}$:⇔
- $Q_0, \ldots, Q_{n+1}, \ldots$ are queries, each empty or with one literal selected in it;
- $\theta_1, \ldots, \theta_{n+1}, \ldots$ are substitutions;
- $c_1, \ldots, c_{n+1}, \ldots$ 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 :⇔

- $\mathcal{T}$ 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 :⇔ it contains a leaf marked as "success"

tree $T \in \mathcal{T}$ finitely failed :⇔ 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 $\mathcal{F} = (\mathcal{T}, T, subs)$:⇔

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

       ⇒ Q is marked as "failure"
     - else

       ⇒ 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 $\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2, \ldots$, 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

:⇔ its main tree is successful

(pre-)SLDNF-tree finitely failed

:⇔ its main tree is finitely failed

SLDNF-tree finite

:⇔ no infinite paths exist in it,

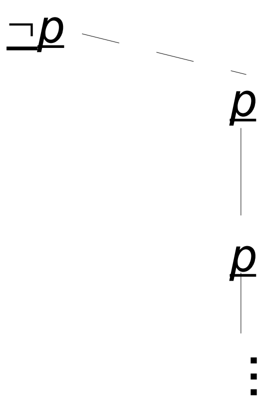
where a path is a sequence of nodes N_0, N_1, N_2, ... such that for every $i = 0, 1, 2, ...$:

- 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:



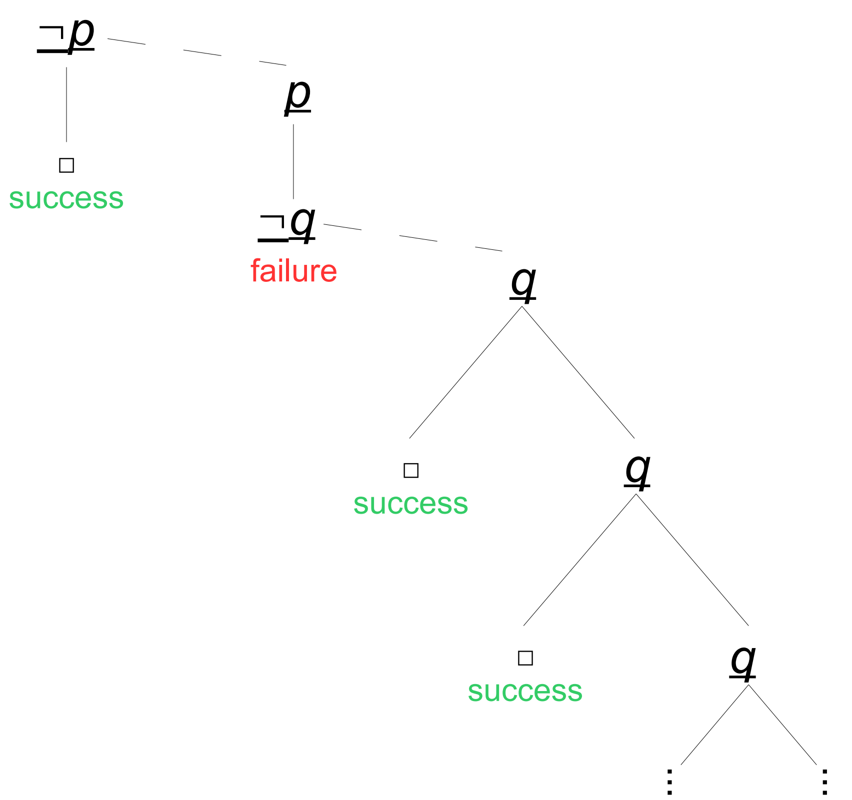
Example (II)

$p \leftarrow \neg q$

$q \leftarrow$

$q \leftarrow q$

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



SLDNF-Derivation

SLDNF-derivation of $P \cup \{Q\}$:⇔

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 :⇔

it ends with □

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

$$\xi = Q_0 \overset{\theta_1}{\Longrightarrow} Q_1 \ldots Q_{n-1} \overset{\theta_n}{\Longrightarrow} Q_n = \square :$$

computed answer substitution (CAS) of Q_0 (w.r.t. ξ) :⇔ $(\theta_1 \cdots \theta_n) \mid_{Var(Q_0)}$

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.

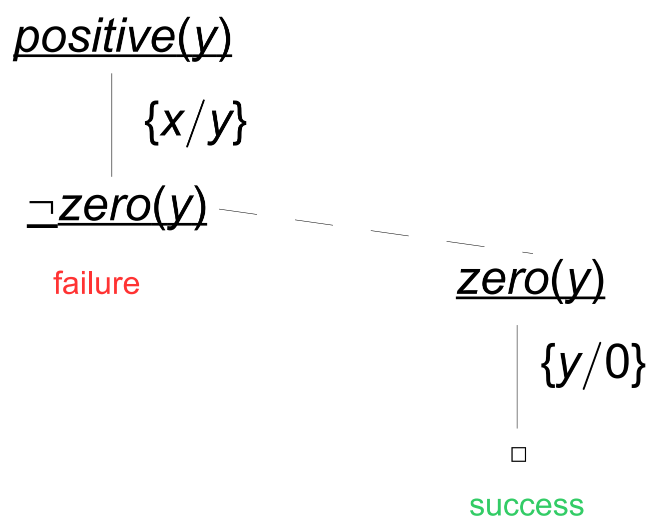
(ii) If the SLDNF-tree $\mathcal{F}$ is the limit of the sequence $\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2, \ldots$, then:

a) $\mathcal{F}$ is successful and yields CAS θ
   iff some $\mathcal{F}_i$ is successful and yields CAS θ,

b) $\mathcal{F}$ finitely failed
   iff some $\mathcal{F}_i$ is finitely failed.

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

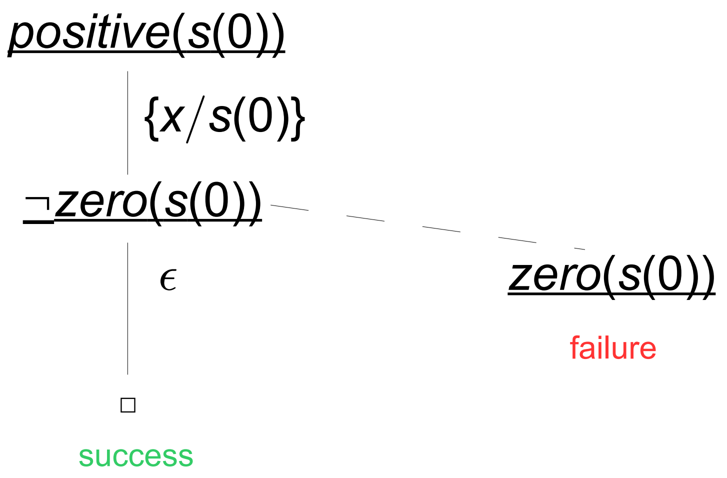
c_1:	$zero(0) \leftarrow$
c_2:	$positive(x) \leftarrow \neg zero(x)$



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

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

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

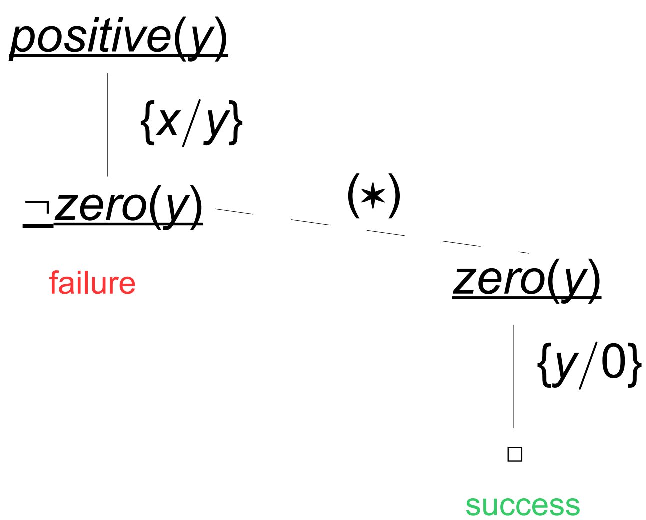


Hence, $positive(s(0))$!, i.e. $\exists y\ positive(y)$!

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 $x \in Var(Q)$ 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 θ is a CAS of Q, then $Q\theta$ is ground.

An Example

$zero(0) \leftarrow$
$positive(x) \leftarrow \neg zero(x)$

This program is not allowed.

$zero(0) \leftarrow$
$positive(x) \leftarrow num(x), \neg zero(x)$
$num(0) \leftarrow$
$num(s(x)) \leftarrow num(x)$

This program is allowed.

Specifics of PROLOG

- Leftmost selection rule
  LDNF-resolution, LDNF-resolvent, LDNF-tree, ...
- Non-ground negative literals are selected!
- A progam 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 $(\{T_{Q_0}\}, T_{Q_0}, subs)$, where T_{Q_0} contains the single node Q_0 and $subs(Q_0)$ is undefined
- Repeatedly apply to current forest $\mathcal{F} = (\mathcal{T}, T, subs)$ 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

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-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

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