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]

Why Negation? Example (I)

\[
\begin{align*}
\text{attend}(\text{flp}, \text{andreas}) & \leftarrow \\
\text{attend}(\text{flp}, \text{maja}) & \leftarrow \\
\text{attend}(\text{flp}, \text{dirk}) & \leftarrow \\
\text{attend}(\text{flp}, \text{natalia}) & \leftarrow \\
\text{attend}(\text{fcp}, \text{andreas}) & \leftarrow \\
\text{attend}(\text{fcp}, \text{maja}) & \leftarrow \\
\text{attend}(\text{fcp}, \text{natalia}) & \leftarrow \\
\text{attend}(\text{fcp}, \text{stefan}) & \leftarrow \\
\text{attend}(\text{fcp}, \text{arturo}) & \leftarrow
\end{align*}
\]

Who attends FCP but not FLP?

\[
\text{attend}(\text{fcp}, x), \neg \text{attend}(\text{flp}, x)
\]
Why Negation? Example (II)

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

$\iff$

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

\[
\text{disjoint}([], x) \leftarrow \\
\text{disjoint}([x|y], z) \leftarrow \neg \text{member}(x, z), \text{disjoint}(y,z)
\]
Extended Logic Programs and Queries

- “¬” negation sign
- $A, \neg A$ literals $\iff A$ atom
- $A, \neg A$ ground literals $\iff A$ ground atom
- (extended) query $\iff$ finite sequence of literals
- $H \leftarrow \mathbf{B}$ (extended) clause
  $\iff H$ atom, $\mathbf{B}$ extended query
- (extended) program
  $\iff$ finite set of extended clauses
How do we Compute?

Negation as Failure ($\text{NF}$) $\iff$

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

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

$\text{SLDNF} = \text{Selection rule driven Linear resolution for Definite clauses augmented by Negation as Failure rule}$
SLDNF-Resolvents

1. $Q = L, A, N$ query; $A$ selected, positive literal
   - $H \leftarrow M$ variant of a clause $c$ which is variable-disjoint with $Q$, $\theta$ $\text{MGU}$ of $A$ and $H$
   - $Q' = (L, M, N)\theta$ SLDNF-resolvent of $Q$ (and $c$ w.r.t. $A$ with $\theta$)
   - We write this SLDNF-derivation step as $Q \Rightarrow_{c}^{\theta} Q'$

2. $Q = L, \neg A, N$ query; $\neg A$ selected, negative ground literal
   - $Q' = L, N$ SLDNF-resolvent of $Q$ (w.r.t. $\neg A$ with $\epsilon$)
   - We write this SLDNF-derivation step as $Q \Rightarrow_{\epsilon}^{\theta} Q'$
Pseudo Derivations

A maximal sequence of SLDNF-derivation steps

\[
Q_0 \Rightarrow_{\theta_1} Q_1 \ldots \Rightarrow_{\theta_{n+1}} Q_n \Rightarrow_{c_{n+1}} Q_{n+1} \ldots
\]

is a pseudo derivation of \( P \cup \{Q_0\} : \Leftarrow \Rightarrow \\

- \( Q_0, \ldots, Q_{n+1}, \ldots \) 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

$\mathcal{F} = (\mathcal{T}, T, subs)$ \text{ forest} \iff

- $\mathcal{T}$ 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 $A$.

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

$T \in \mathcal{T}$ finitely failed $\iff$ 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 ($\{T_Q\}, T_Q, \text{subs}$) is in $\mathcal{C}$, where $T_Q$ contains the single node $Q$ and $\text{subs}(Q)$ 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 $\mathcal{F} = (T, \: T, \: \text{subs}) : \Leftrightarrow$

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 $T$, 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 $T$ 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
\[ \iff \text{limit of a sequence } \mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2, \ldots, \text{ where} \]

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

SLDNF-tree for \( P \cup \{Q\} \)
\[ \iff \]
SLDNF-tree in which \( Q \) is the root of the main tree
Successful, Failed, and Finite SLDNF-Trees

(pre-)SLDNF-tree successful
:\iff its main tree is successful

(pre-)SLDNF-tree finitely failed
:\iff its main tree is finitely failed

SLDNF-tree finite
:\iff no infinite paths exist in it,
where a path is a sequence of nodes $N_0, N_1, N_2, \ldots$ such that for every $i = 0, 1, 2, \ldots$:
\begin{itemize}
  \item either $N_{i+1}$ is a direct descendant of $N_i$
  \item or $N_{i+1}$ is the root of $subs(N_i)$.
\end{itemize}
Example (I)

\[ p \leftarrow p \]

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

\[
\begin{array}{c}
\neg p \\
\quad \neg p \\
\quad \quad p \\
\quad \quad \quad p \\
\quad \quad \quad \vdots
\end{array}
\]
Example (II)

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

$p \leftarrow \neg q$
$q \leftarrow$
$q \leftarrow q$
SLDNF-Derivation

SLDNF-derivation of $P \cup \{Q\} \iff$

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 $\iff$

it ends with $\square$

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

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

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

$$(\theta_1 \ldots \theta_n) \mid_{\text{Var}(Q_0)}$$
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 $\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2, \ldots$, then:

a) $\mathcal{F}$ is successful and yields $\mathsf{CAS} \, \theta$

   iff some $\mathcal{F}_i$ is successful and yields $\mathsf{CAS} \, \theta$,

b) $\mathcal{F}$ finitely failed

   iff some $\mathcal{F}_i$ is finitely failed.
Why Only Select Negative Literals if they are Ground? (I)

\[ \begin{align*}
  c_1: & \quad \text{zero}(0) \leftarrow \\
  c_2: & \quad \text{positive}(x) \leftarrow \neg \text{zero}(x)
\end{align*} \]

\[
\begin{array}{c}
\text{positive}(y) \\
\downarrow \{x/y\} \\
\neg \text{zero}(y) \\
\downarrow \text{failure} \\
\neg \text{zero}(y) \\
\downarrow \\
\text{zero}(y) \\
\downarrow \{y/0\} \\
\square \\
\text{success}
\end{array}
\]

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)

\[ c_1: \quad \text{zero}(0) \leftarrow \]
\[ c_2: \quad \text{positive}(x) \leftarrow \neg \text{zero}(x) \]

\[ \text{positive}(s(0)) \]
\[ \{x/s(0)\} \]
\[ \neg \text{zero}(s(0)) \]
\[ \epsilon \]
\[ \square \]
\[ \text{success} \]
\[ \neg \text{zero}(s(0)) \]
\[ \text{failure} \]

Hence, \( \text{positive}(s(0))! \), i.e. \( \exists y \text{ positive}(y)! \)
Why Only Select Negative Literals if they are Ground? (III)

\[ c_1: \quad \text{zero}(0) \leftarrow \]
\[ c_2: \quad \text{positive}(x) \leftarrow \neg \text{zero}(x) \]

\[
\begin{align*}
\text{positive}(y) & \rightarrow \{x/y\} \\
\neg \text{zero}(y) & \rightarrow (\ast) \\
\text{failure} & \rightarrow \neg \text{zero}(y) \\
\text{success} & \rightarrow \text{zero}(y) \rightarrow \{y/0\}
\end{align*}
\]

Fundamental mistake in (\ast): \( \exists y \neg \text{zero}(y) \) is not the opposite of \( \exists y \neg \neg \text{zero}(y) \)
Selection of Non-Ground Negative Literals in Prolog

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

| ?- positive(0).      no
| ?- positive(s(0)).   yes
| ?- positive(Y).      no
```
Extended Selection Rules

(extended) selection rule :⇔

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

SLDNF-tree \( \mathcal{F} \) is according to selection rule \( \mathcal{R} \) :⇔

\( \mathcal{F} \) 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 :⇔

\( \mathcal{R} \) never selects a non-ground negative literal
Blocked Queries

query $Q$ blocked
\[
\iff
\]
$Q$ non-empty and contains exclusively non-ground negative literals

$P \cup \{Q\}$ flounders
\[
\iff
\]
some SLDNF-tree for $P \cup \{Q\}$ contains a blocked node
Allowed Programs and Queries

query $Q$ allowed  
$\iff$
\begin{align*}
\text{every } x \in Var(Q) \text{ occurs in a positive literal of } Q
\end{align*}

clause $H \leftarrow B$ allowed $\iff \neg H, B$ allowed

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

program $P$ allowed $\iff$ 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

\[
\begin{align*}
\text{zero}(0) & \leftarrow \\
\text{positive}(x) & \leftarrow \neg \text{zero}(x)
\end{align*}
\]

This program is not allowed.

\[
\begin{align*}
\text{zero}(0) & \leftarrow \\
\text{positive}(x) & \leftarrow \text{num}(x), \neg \text{zero}(x) \\
\text{num}(0) & \leftarrow \\
\text{num}(s(x)) & \leftarrow \text{num}(x)
\end{align*}
\]

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 $\mathcal{F} = (\mathcal{I}, T, \text{subs})$ and leftmost unmarked leaf $Q$ in $T_1$, where $T_1 \in \mathcal{I}$ is leftmost, bottommost (=most nested subsidiary) tree with an unmarked leaf, the operation $\text{expand}(\mathcal{F}, Q)$
Operation Expand

operation \( \text{expand}(\mathcal{F}, Q) \) 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):
    \( \star \) if \( \text{subs}(Q) \) is undefined, then add a new tree \( T' = A \) and set \( \text{subs}(Q) \) to \( T' \)
    \( \star \) if \( \text{subs}(Q) \) is defined and successful, then mark \( Q \) with “failure”
    \( \star \) if \( \text{subs}(Q) \) 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)

\[
\text{even}(0).
\text{even}(X) \leftarrow \neg \text{odd}(X).
\text{odd}(s(X)) \leftarrow \text{even}(X).
\]

\[
| \ ?- \text{even}(X).
\]

\[
x = 0 ;
\]

\[
\text{no}
\]

\[
| \ ?- \text{even}(s(s(0))).
\]

\[
\text{yes}
\]
Floundering is Ignored (II)

\[\begin{align*}
\text{num}(0). \\
\text{num}(s(X)) & : \text{num}(X). \\
\text{even}(X) & : \text{num}(X), \neg \text{odd}(X). \\
\text{odd}(s(X)) & : \text{even}(X). \\
\end{align*}\]

\[\begin{align*}
\text{?- even}(X). \\
X = 0 ; \\
X = s(s(0)) ; \\
X = s(s(s(s(0)))) ; \\
\vdots
\end{align*}\]
Objectives

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