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)

```
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) $A = [a_1, ..., a_m]$ and $B = [b_1, ..., b_n]$ disjoint

$\iff$

- $m = 0$, or
- $m > 0$, $a_1 \notin B$, and $[a_2, ..., a_m]$ 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
- $H \leftarrow B$ (extended) clause $\iff$ $H$ atom, $B$ 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 $Q = L, \neg A, N$.
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 $Q' = L, N$.

$\neg A$ 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 $Q$, $\theta$ 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 $Q \xrightarrow{\theta} Q'$

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 $Q \xrightarrow{\epsilon} Q'$
Pseudo Derivations

A maximal sequence of SLDNF-derivation steps

\[ Q_0 \Rightarrow^{\theta_1} Q_1 \ldots \Rightarrow^{\theta_{n+1}} Q_n \Rightarrow^{c_1} Q_{n+1} \ldots \]

- 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

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

- $\text{subs}$ 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 $\iff$ it contains a leaf marked as “success”
tree $T \in \mathcal{T}$ finitely failed $\iff$ it is finite and all leaves are marked as “failure”
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 $\mathcal{F} = (T, T, \text{subs})$ :

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

SLDNF-tree

\[ \iff \text{limit of a sequence } F_0, F_1, F_2, \ldots, \text{ where} \]

- \( F_0 \) initial pre-SLDNF-tree
- \( F_{i+1} \) extension of \( F_i \), 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**
\[\iff \text{its main tree is successful}\]

(pre-)SLDNF-tree **finitely failed**
\[\iff \text{its main tree is finitely failed}\]

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

\[ p \leftarrow p \]

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

```
\neg p
  \neg p
    \neg p
      p
        p
          p
            \vdots
```
Example (II)

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

$\neg p$  
\hline
success

$p$  
\hline
$q$  
\hline
failure

$q$  
\hline
\hline
$q$  
\hline
success

$q$  
\hline
\hline
$q$  
\hline
success

\vdots
\vdots
SLDNF-Derivation

SLDNF-derivation of $P \cup \{Q\}$ :⇔

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

SLDNF-derivation successful :⇔

it ends with $\Box$

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

$$\xi = Q_0 \stackrel{\theta_1}{\longrightarrow} Q_1 \ldots Q_{n-1} \stackrel{\theta_n}{\longrightarrow} Q_n = \Box :$$

computed answer substitution (CAS) of $Q_0$ (w.r.t. $\xi$) :⇔ $(\theta_1 \cdots \theta_n) \mid_{\text{Var}(Q_0)}$
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 $\text{CAS } \theta$ if some $\mathcal{F}_i$ is successful and yields $\text{CAS } \theta$,

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

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

\[
\text{positive}(y) \\
\{x/y\} \\
\neg \text{zero}(y) \\
\text{failure} \\
\neg \text{zero}(y) \\
\text{success}
\]

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)

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

\[
\begin{array}{c}
\text{positive}(s(0)) \\
\quad \{x/s(0)\} \\
\neg \text{zero}(s(0)) \\
\quad \epsilon \\
\end{array}
\]

\[
\text{zero}(s(0)) \\
\text{failure}
\]

\[
\text{success}
\]

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

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

Fundamental mistake in (\(*\)): \(\exists y \text{ zero}(y)\) is not the opposite of \(\exists y \neg \text{zero}(y)\)
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 rule :\iff

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

SLDNF-tree $\mathcal{F}$ is according to selection rule $\mathcal{R}$ :\iff
$\mathcal{F}$ 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 :\iff
$\mathcal{R}$ never selects a non-ground negative literal
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
⇔
every $x \in Var(Q)$ occurs in a positive literal of $Q$

clause $H \leftarrow B$ allowed $:\Leftrightarrow \neg H, B$ 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) ←
um(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
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 $\mathcal{F} = (T, T, \text{subs})$ and leftmost unmarked leaf $Q$ in $T_1$, where $T_1 \in T$ 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):
    - if $\text{subs}(Q)$ is undefined, then add a new tree $T' = A$ and set $\text{subs}(Q)$ to $T'$
    - if $\text{subs}(Q)$ is defined and successful, then mark $Q$ with “failure”
    - 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)

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(0))) ;
::
Objectives

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