Dresden University of Technology

Computational Logic Group

Hannes Strass

(based on slides by Michael Thielscher)

Faculty of Computer Science, Institute of Artificial Intelligence, Computational Logic Group

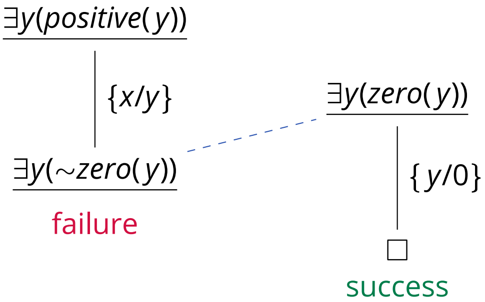
Soundness and “Completeness” of SLDNF Resolution

Lecture 8, 1st Dec 2025 // Foundations of Logic Programming, WS 2025/26

Previously ...

- **Normal logic programs** allow for “negation” in queries (clause bodies).
- The **negation as failure** rule treats negated atoms $\sim A$ in queries by asking the query A in a **subsidiary tree** and negating the answer.
- A proof theory for normal logic programs is given by **SLDNF resolution**.
- Care must be taken not to let **non-ground negative literals** get selected.
- A clause is **safe** iff each of its variables occurs in a positive body literal.

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



Overview

First-Order Formulas and Logical Truth

Completion of Programs

Soundness of SLDNF Resolution

Restricted Completeness of SLDNF Resolution

First-Order Formulas and Logical Truth

First-Order Formulas

Definition

Let Π and F be ranked alphabets of predicate symbols and function symbols, respectively, and V be a set of variables.

The set of **(first-order) formulas** (over Π, F, and V) is inductively defined as follows:

- if atom $A \in TB_{\Pi,F,V}$, then A is a formula;
- if G_1 and G_2 are formulas, then $\neg G_1$, $G_1 \wedge G_2$ (also written G_1, G_2), $G_1 \vee G_2$, $G_1 \leftarrow G_2$, and $G_1 \leftrightarrow G_2$ are formulas;
- if G is a formula and $x \in V$, then $\forall x G$ and $\exists x G$ are formulas.

Note

Whenever we interpret normal queries/clauses as first-order formulas, we (for now) take $\sim$ to be $\neg$.

Extended Notion of Logical Truth (1)

Definition

Let G be a formula, I be an interpretation with domain D, and $\sigma\colon V \to D$ be a state.

Formula G **is true in** I **under** σ, written $I, \sigma \Vdash G$, based on the structure of G:

- $I, \sigma \Vdash p(t_1, \ldots, t_n) \quad :\Longleftrightarrow \quad (\sigma(t_1), \ldots, \sigma(t_n)) \in p_I$
- $I, \sigma \Vdash \neg G \quad :\Longleftrightarrow \quad I, \sigma \nVdash G$
- $I, \sigma \Vdash G_1 \land G_2 \quad :\Longleftrightarrow \quad I, \sigma \Vdash G_1$ and $I, \sigma \Vdash G_2$
- $I, \sigma \Vdash G_1 \lor G_2 \quad :\Longleftrightarrow \quad I, \sigma \Vdash G_1$ or $I, \sigma \Vdash G_2$
- $I, \sigma \Vdash G_1 \leftarrow G_2 \quad :\Longleftrightarrow \quad$ if $I, \sigma \Vdash G_2$ then $I, \sigma \Vdash G_1$
- $I, \sigma \Vdash G_1 \leftrightarrow G_2 \quad :\Longleftrightarrow \quad I, \sigma \Vdash G_1$ iff $I, \sigma \Vdash G_2$
- $I, \sigma \Vdash \forall x G \quad :\Longleftrightarrow \quad$ for every $d \in D$: $I, \sigma[x \mapsto d] \Vdash G$
- $I, \sigma \Vdash \exists x G \quad :\Longleftrightarrow \quad$ for some $d \in D$: $I, \sigma[x \mapsto d] \Vdash G$

where $\sigma[x \mapsto d]$ denotes $\sigma'\colon V \to D$ with $\sigma'(x) = d$ and $\sigma'(y) = \sigma(y)$ for $y \in V \setminus \{x\}$.

Extended Notion of Logical Truth (2)

Definition

Let G be a formula, S and T be sets of formulas, and I be an interpretation. Furthermore, let $x_1, \ldots, x_k$ be the variables occurring in G.

- $\forall x_1, \ldots, \forall x_k G$ is the **universal closure** of G (abbreviated $\forall G$).
- $I \Vdash \forall G \;\; :\Longleftrightarrow \;\; I, \sigma \Vdash G$ for every state σ
- **G is true in I** (or: **I is a model of G**), written: $I \Vdash G \;\; :\Longleftrightarrow \;\; I \Vdash \forall G$
- **I is a model of S**, written: $I \Vdash S \;\; :\Longleftrightarrow \;\; I \Vdash G$ for every $G \in S$
- **T is a semantic (or: logical) consequence of S**, written: $S \models T$
  $:\Longleftrightarrow$ every model of S is a model of T

Negative Consequences of Logic Programs (1)

Consider program P_{mem}:

$$member(x, [x \mid y]) \leftarrow$$
$$member(x, [y \mid z]) \leftarrow member(x, z)$$

Then, e.g. $P_{mem} \models member(a, [a, b])$ and $P_{mem} \not\models member(a, [\,])$.

But also $P_{mem} \not\models \neg member(a, [\,])$, since $HB_{\{member\},\{\mid,[\,],a\}} \Vdash P_{mem}$ and $HB_{\{member\},\{\mid,[\,],a\}} \nVdash \neg member(a, [\,])$.

Nevertheless the SLDNF tree of $P_{mem} \cup \{\sim member(a, [\,])\}$ is successful:

$$\underline{\sim member(a, [\,])}$$
success

$$\underline{member(a, [\,])}$$
failure

Negative Consequences of Logic Programs (2)

Observation

For every normal logic program P over vocabulary Π, F, the Herbrand base $HB_{\Pi,F}$ is a model of P. (All implications are satisfied.)

Corollary

There is no negative ground literal $\neg A$ that is a logical consequence of P.

But: The SLDNF tree of $P \cup \{\sim A\}$ may be successful!

$\rightsquigarrow$ Taking $\sim$ to be $\neg$, SLDNF resolution is not sound w.r.t. $P \models$.

Solution:

Avoid Herbrand models that are "too large".

$\rightsquigarrow$ Formalise "the information in the program is all there is".

$\rightsquigarrow$ Strengthen P to its completion $comp(P)$.

Completion of Programs

Completed Definitions (Example 1)

$$P: \quad \begin{array}{lcl} \mathit{happy} & \leftarrow & \mathit{sun}, \mathit{holidays} \\ \mathit{happy} & \leftarrow & \mathit{snow}, \mathit{holidays} \\ \mathit{snow} & \leftarrow & \mathit{cold}, \mathit{precipitation} \\ \mathit{cold} & \leftarrow & \mathit{winter} \\ \mathit{precipitation} & \leftarrow & \mathit{holidays} \\ \mathit{winter} & \leftarrow & \\ \mathit{holidays} & \leftarrow & \end{array}$$

$$\mathit{comp}(P): \quad \begin{array}{lcl} \mathit{happy} & \leftrightarrow & (\mathit{sun} \land \mathit{holidays}) \lor (\mathit{snow} \land \mathit{holidays}) \\ \mathit{snow} & \leftrightarrow & \mathit{cold} \land \mathit{precipitation} \\ \mathit{cold} & \leftrightarrow & \mathit{winter} \\ \mathit{precipitation} & \leftrightarrow & \mathit{holidays} \\ \mathit{winter} & \leftrightarrow & \mathit{true} \\ \mathit{holidays} & \leftrightarrow & \mathit{true} \\ \mathit{sun} & \leftrightarrow & \mathit{false} \end{array}$$

Then, $\mathit{comp}(P) \models \mathit{happy}, \mathit{snow}, \mathit{cold}, \mathit{precipitation}, \mathit{winter}, \mathit{holidays}, \neg \mathit{sun}$.

Completed Definitions (Example 2)

$$P: \quad \begin{array}{lcl} member(x,[x\,|\,y]) & \leftarrow & \\ member(x,[y\,|\,z]) & \leftarrow & member(x,z) \\ disjoint([\,],x) & \leftarrow & \\ disjoint([x\,|\,y],z) & \leftarrow & \sim member(x,z),\ disjoint(y,z) \end{array}$$

$$comp(P): \quad \begin{array}{lcl} \forall x_1,x_2 \Big(member(x_1,x_2) & \leftrightarrow & (\exists x,y\,(x_1 = x \wedge x_2 = [x\,|\,y]) \vee \\ & & \exists x,y,z\,(x_1 = x \wedge x_2 = [y\,|\,z] \wedge member(x,z))) \Big) \\ \forall x_1,x_2 \Big(disjoint(x_1,x_2) & \leftrightarrow & (\exists x\,(x_1 = [\,] \wedge x_2 = x) \vee \\ & & \exists x,y,z\,(x_1 = [x\,|\,y] \wedge x_2 = z \wedge \\ & & \quad \neg member(x,z) \wedge disjoint(y,z))) \Big) \end{array}$$

plus standard equality and inequality axioms

Then, e.g. $comp(P) \models member(a,[a,b])$, $\neg member(a,[\,])$, $\neg disjoint([a],[a])$.

Completion (1)

Definition (Clark, 1978)

Let P be a normal logic program. The **completion** of P (denoted by $comp(P)$) is the set of formulas constructed from P by the following 6 steps:

1. Associate with every n-ary predicate symbol p (from P) a sequence of pairwise distinct variables $x_1, \ldots, x_n$ that do not occur in P.

2. Transform each clause $c = p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_k$ into
$$p(x_1, \ldots, x_n) \leftarrow x_1 = t_1 \wedge \ldots \wedge x_n = t_n \wedge B_1 \wedge \ldots \wedge B_k$$
Any empty conjunction (for $n = 0$ or $k = 0$) is replaced by *true*.

3. Transform each resulting formula $p(x_1, \ldots, x_n) \leftarrow G$ into
$$p(x_1, \ldots, x_n) \leftarrow \exists \vec{z}\, G$$
where $\vec{z}$ is a sequence of the elements of $Var(c)$.

Completion (2)

Definition (Clark, 1978, continued)

4. For every n-ary predicate symbol p, let

$$p(x_1,\ldots,x_n) \leftarrow \exists\vec{z}_1 G_1, \quad \ldots, \quad p(x_1,\ldots,x_n) \leftarrow \exists\vec{z}_m G_m$$

be all implications obtained in Step 3 ($m \geq 0$).

If $m > 0$, then replace these by the formula

$$\forall x_1,\ldots,x_n \Big(p(x_1,\ldots,x_n) \leftrightarrow (\exists\vec{z}_1 G_1 \vee \ldots \vee \exists\vec{z}_m G_m) \Big)$$

If $m = 0$, then add the formula

$$\forall x_1,\ldots,x_n (p(x_1,\ldots,x_n) \leftrightarrow \mathit{false})$$

Completion (3)

Definition (Clark, 1978, continued)

5. Add the **standard axioms of equality**:

$$\begin{array}{ll} \forall\ [\ x = x &] \\ \forall\ [\ x = y\ \rightarrow\ y = x &] \\ \forall\ [\ x = y \land y = z\ \rightarrow\ x = z &] \\ \forall\ [\ x_i = y\ \rightarrow\ f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, y, \ldots, x_n) &] \\ \forall\ [\ x_i = y\ \rightarrow\ (p(x_1, \ldots, x_i, \ldots, x_n) \leftrightarrow p(x_1, \ldots, y, \ldots, x_n)) &] \end{array}$$

6. Add the **standard axioms of inequality**:

$$\forall\ [\ x_1 \neq y_1 \lor \ldots \lor x_n \neq y_n\ \rightarrow\ f(x_1, \ldots, x_n) \neq f(y_1, \ldots, y_n)\ ]$$

$$\forall\ [\ f(x_1, \ldots, x_m) \neq g(y_1, \ldots, y_n)\ ] \qquad \text{(whenever } f \neq g\text{)}$$

$$\forall\ [\ x \neq t\ ] \qquad \text{(whenever } x \text{ is a proper subterm of } t\text{)}$$

Steps 5. and 6. ensure that "$=$" must be interpreted as equality.

Quiz: Completion

Quiz

Consider the following logic program P: …

Soundness of SLDNF Resolution

Soundness of SLDNF Resolution

Definition

Let P be a normal logic program, Q be a normal query, and θ be a substitution.

- $\theta|_{Var(Q)}$ is a **correct answer substitution** of Q $:\Longleftrightarrow$ $comp(P) \models Q\theta$
- $Q\theta$ is a **correct instance** of Q $:\Longleftrightarrow$ $comp(P) \models Q\theta$

Theorem (Lloyd, 1987)

If there exists a successful SLDNF derivation of $P \cup \{Q\}$ with computed answer substitution θ, then $comp(P) \models Q\theta$.

Corollary

If there exists a successful SLDNF derivation of $P \cup \{Q\}$, then $comp(P) \models \exists Q$.

We assume here that $\sim$ in queries has been replaced by $\neg$ for entailment.

Restricted Completeness of SLDNF Resolution

Incompleteness (1): Inconsistency

$$P: \quad p \leftarrow \sim p$$

By definition, it follows that $comp(P) \supseteq \{p \leftrightarrow \neg p\} \equiv \{false\}$.

Hence, $comp(P) \models p$ and $comp(P) \models \neg p$ because $I \not\Vdash comp(P)$ for every interpretation I, i.e. $comp(P)$ has no model ($comp(P)$ is **unsatisfiable**).

But there is neither a successful SLDNF derivation of $P \cup \{p\}$ nor of $P \cup \{\sim p\}$.

Incompleteness (2): Non-Strictness

$$P: \quad \begin{array}{rcl} p & \leftarrow & q \\ p & \leftarrow & \sim q \\ q & \leftarrow & q \end{array}$$

Thus $comp(P) \supseteq \{p \leftrightarrow (q \vee \neg q), \quad q \leftrightarrow q\} \equiv \{p \leftrightarrow true\}$.

Hence, $comp(P) \models p$.

But there is no successful SLDNF derivation of $P \cup \{p\}$.

Incompleteness (3): Floundering

$$P: \quad p(x) \leftarrow {\sim}q(x)$$

Thus $comp(P) \supseteq \{\forall x_1(p(x_1) \leftrightarrow \exists x(x_1 = x \wedge \neg q(x))), \quad \forall x_1(q(x_1) \leftrightarrow \mathit{false})\}$
$\equiv \{\forall x_1(p(x_1) \leftrightarrow \mathit{true}), \quad \forall x_1(q(x_1) \leftrightarrow \mathit{false})\}.$

Hence, $comp(P) \models \forall x_1 p(x_1)$.

But there is no successful SLDNF derivation of $P \cup \{p(x_1)\}$.

Incompleteness (4): Unfairness

$$\begin{array}{llll} P: & r & \leftarrow & p, q \\ & p & \leftarrow & p \end{array}$$

$$\begin{aligned} \text{Thus } \mathit{comp}(P) &\supseteq \{r \leftrightarrow (p \wedge q), \quad p \leftrightarrow p, \quad q \leftrightarrow \mathit{false}\} \\ &\equiv \{r \leftrightarrow \mathit{false}, \quad q \leftrightarrow \mathit{false}\}. \end{aligned}$$

Hence, $\mathit{comp}(P) \models \neg r$.

But there is no successful SLDNF derivation of $P \cup \{\sim r\}$ w.r.t. leftmost selection rule.

Dependency Graphs

Definition

The **dependency graph** D_P of a normal logic program P is a directed graph with labelled edges, where

- the nodes are the predicate symbols of P
- the edges are either labelled by $+$ (positive edge) or by $-$ (negative edge)
- there is an edge $q \xrightarrow{+} p$ in D_P
  $:\iff \quad P$ contains a clause $p(s_1, \ldots, s_m) \leftarrow \vec{L}, q(t_1, \ldots, t_n), \vec{N}$
- there is an edge $q \xrightarrow{-} p$ in D_P
  $:\iff \quad P$ contains a clause $p(s_1, \ldots, s_m) \leftarrow \vec{L}, {\sim}q(t_1, \ldots, t_n), \vec{N}$

Note that the direction of the edges is sometimes reversed in other texts.

Strict, Hierarchical, Stratified Programs

Definition

Let P be a normal program with dependency graph D_P,
let p, q be predicate symbols, and Q be a normal query.

- p depends **evenly** (resp. **oddly**) on q
  $:\Longleftrightarrow$ there is a path from q to p in D_P with
  an even (resp. odd) number of *negative* edges
- P is **strict** w.r.t. Q
  $:\Longleftrightarrow$ no predicate symbol occurring in Q depends both evenly and oddly on a predicate symbol in the head of a clause in P
- P is **hierarchical** $:\Longleftrightarrow$ no cycle exists in D_P
- P is **stratified** $:\Longleftrightarrow$ no cycle with a negative edge exists in D_P

“Completeness” of SLDNF Resolution

- Recall: A query is *safe* iff each of its variables occurs in some of its positive atoms.
- Recall: A selection rule is *safe* iff it never selects a non-ground negative literal.

Theorem (Lloyd, 1987)

Fix $\mathcal{R}$ to be a safe selection rule.

Let P be a hierarchical and safe program, and Q be a safe query.

If $comp(P) \models Q\theta$ for some θ such that $Q\theta$ is ground, then there exists a successful SLDNF derivation (via $\mathcal{R}$) of $P \cup \{Q\}$ with cas θ.

Theorem (Cavedon and Lloyd, 1989)

Fix $\mathcal{R}$ to be a safe and fair selection rule.

Let P be a stratified and safe program and Q be a safe query, such that P is strict w.r.t. Q.

If $comp(P) \models Q\theta$ for some θ such that $Q\theta$ is ground, then there exists a successful SLDNF derivation of $P \cup \{Q\}$ with cas θ.

Fair Selection Rules

Definition

An (SLDNF) selection rule $\mathcal{R}$ is **fair**

$:\Longleftrightarrow$ for every SLDNF tree $\mathcal{F}$ via $\mathcal{R}$ and for every branch ξ in $\mathcal{F}$:

- either ξ is failed,
- or for every literal L occurring in a query of ξ, (some further instantiated version of) L is selected within a finite number of derivation steps.

Example

- The selection rule "select leftmost literal" is unfair.
- The selection rule "select leftmost literal to the right of the literals introduced at the previous derivation step, if it exists, otherwise select leftmost literal" is fair.

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 is done without occur check.
- Derivation construction via depth-first search with backtracking.

Extended Prolog Trees

Definition

Let P be a normal program and Q_0 be a normal query.

The **Extended Prolog Tree** for $P \cup \{Q_0\}$ is a forest of finitely branching, ordered 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 the **leftmost**, **bottommost** (most nested subsidiary) tree with an unmarked leaf, the operation *expand*$(\mathcal{F}, Q)$.

Operation Expand

Definition

The operation *expand*$(\mathcal{F}, Q)$ for query Q in tree T_1 is defined as follows:

- if $Q = \Box$, 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 (respecting the order of the clauses in the program)
  - $L = \sim A$ 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 (1)

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

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

```
X = 0 ;
no
```

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

```
yes
```

Floundering is Ignored (2)

```
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)))) ;
```

⋮

Conclusion

Summary

- For every normal logic program P, its **completion** $comp(P)$ replaces the logical implications of clauses by equivalences (to disjunctions of bodies).
- SLDNF resolution w.r.t. P is **sound** for entailment w.r.t. $comp(P)$.
- SLDNF resolution is only **complete** (for entailment w.r.t. $comp(P)$) for certain combinations of classes of programs, queries, and selection rules.
- For a normal program P, its **dependency graph** D_P explicitly shows positive and negative dependencies between predicate symbols.
- A normal program P is **stratified** iff D_P has no cycle with a negative edge.

Suggested action points:

- Construct the completion of the programs on slides 31 and 32.
- Find a program that shows unfairness of the leftmost selection rule.